HEWLETT-PACKARD

ESR

HEWLETT PACKARD

HP 3588A HP-IB Programming Reference

Printed in U.S.A.

Made in U.S.A.

HP 3588A HP-IB Programming Reference

HEWLETT PACKARD

HP Part Number 03588-90025
Microfiche Part Number 03588-90225
Printed in U.S.A.

Print Date: April 1990

HEWLETT PACKARD

SAFETY SUMMARY

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements. This is a Safety Class 1 instrument.

GROUND THE INSTRUMENT

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

DO NOT OPERATE IN AN EXPLOSIVE ATMOSPHERE

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

KEEP AWAY FROM LIVE CIRCUITS

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

DO NOT SERVICE OR ADJUST ALONE

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

DO NOT SUBSTITUTE PARTS OR MODIFY INSTRUMENT

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure the safety features are maintained.

DANGEROUS PROCEDURE WARNINGS

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

Warning **Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.**

HEWLETT PACKARD

SAFETY SYMBOLS

General Definitions of Safety Symbols Used On Equipment or In Manuals.

⚠	Instruction manual symbol: the product will be marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.
ϟ	Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be so marked.)
⏚ OR (⏚)	Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating equipment.
(⏚)	Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual, and before operating the equipment.
⏛ OR ⊥	Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.
∼	Alternating current (power line.)
⎓	Direct current (power line.)
≂	Alternating or direct current (power line.)

Warning — **The WARNING sign denotes a hazard. It calls attention to a procedure, practice, condition or the like, which if not correctly performed or adhered to, could result in injury or death to personnel.**

Caution — The CAUTION sign denotes a hazard. It calls attention to an operating procedure, practice, condition or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.

Note — The NOTE sign denotes important information. It calls attention to procedure, practice, condition or the like, which is essential to highlight.

Table of Contents

Programming Fundamentals

Programming Examples

Command Reference

Introduction to HP-IB

Notice to Experienced HP-IB Programmers

The HP 3588A's HP-IB command set is derived from the TMSL standard (described later in this chapter). TMSL command sets differ from more traditional HP-IB command sets in the following ways:

- A traditional HP-IB command typically consists of a single mnemonic. A TMSL command typically consists of a series of mnemonics separated by colons. The mnemonics are selected from a command hierarchy, which organizes commands into related groups. These multi-mnemonic commands are less cryptic than single-mnemonic commands and help to make your programs more self-documenting. Chapter 3 tells you how to use the TMSL command hierarchy.

- A traditional HP-IB command set contains mnemonics that correspond directly to an instrument's front-panel keys. The HP 3588A's TMSL command set does give you HP-IB access to all front-panel functions, but there is not a one-to-one correspondence between commands and keys. (This results from the fact that the TMSL command hierarchy is organized differently than the front-panel key hierarchy.) A special feature allows the HP 3588A to echo equivalent TMSL command mnemonics when you press a series of front-panel keys. You can enable this feature under the [**Local/HP-IB**] hardkey.

Manual Overview

This manual is organized into five major parts:

- Programming Fundamentals.
- Programming Examples.
- Command Reference.
- Appendices.
- Index.

Programming Fundamentals

This part of the manual contains five chapters, each of which discusses some aspect of programming the HP 3588A via the HP-IB:

- Chapter 1 introduces you to HP-IB and TMSL concepts. It also tells you how to configure the HP 3588A in an HP-IB system.
- Chapter 2 tells you how the analyzer interacts with the controller and other devices on the HP-IB.
- Chapter 3 describes the TMSL command hierarchy.
- Chapter 4 tells you how data is transferred between the analyzer and a controller.
- Chapter 5 describes the analyzer's register structure and tells you how the analyzer uses registers to generate service requests.

Programming Examples

This part of the manual contains commented programming examples. It may be a good place to start if you are an experienced HP-IB programmer and are already familiar with TMSL concepts.

Command Reference

This part of the manual contains a detailed description of each HP-IB command. The command descriptions are organized alphabetically.

Appendices

This part of the manual contains two appendices:

- Appendix A provides a quick reference to the HP 3588A's HP-IB command set.
- Appendix B provides a complete listing of the HP 3588A's error messages.

Index

This part of the manual references the page numbers where different subjects are discussed. It can be especially useful for determining which command you should use to access a particular analyzer function.

HP-IB Overview

What is HP-IB?

HP-IB—the Hewlett-Packard Interface Bus—is a high-performance bus that allows you to build integrated test systems from individual instruments and computers. The bus and its associated interface operations are defined by the IEEE 488.1 standard (described later in this chapter).

HP-IB cables provide the physical link between devices on the bus. There are eight data lines on each cable that are used to send data from one device to another. Devices that can be addressed to send data over these lines are called "talkers," and those that can be addressed to receive data are called "listeners." There are also five control lines on each cable that are used to manage traffic on the data lines and to control other interface operations. Devices that can use these control lines to specify the talker and listener in a data exchange are called "controllers."

When an HP-IB system contains more than one device with controller capabilities, only one of the devices is allowed to control data exchanges at any given time. The device currently controlling data exchanges is called the "active controller." Also, only one of the controller-capable devices can be designated as the "system controller." The system controller is the one device that can take control of the bus even if it is not the active controller. The HP 3588A can act as a talker, listener, active controller, or system controller at different times.

HP-IB addresses provide a way to identify devices on the bus. For example, the active controller uses HP-IB addresses to specify which device talks and which device listens during a data exchange. This means that each device's address must be unique. You set a device's address on the device itself, usually using a rear-panel switch or a front-panel key sequence.

Sending Commands Over the HP-IB

Commands are sent over the HP-IB via a controller's language system, such as BASIC or Pascal. As a result, you will need to determine which keywords your controller's language system uses to send HP-IB commands. When looking for keywords, keep in mind that there are actually two different kinds of HP-IB commands:

- *Bus management commands*, which control the HP-IB interface.
- *Device commands*, which control analyzer functions.

Language systems usually deal differently with these two kinds of HP-IB commands. For example, HP BASIC uses a unique keyword to send each bus management command, but always uses the keyword OUTPUT to send device commands. For more information on the differences between bus management commands and device commands, see chapter 2, "Behavior in an HP-IB System."

The following example shows how to send a typical device command:

```
OUTPUT 719;"AVERAGE:COUNT 5"
```

This sends the command within the quotes (AVERAGE:COUNT 5) to the HP-IB device at address 719. If the device is an HP 3588A, the command instructs the analyzer to set the number of averages to 5.

Note

All examples in this manual are written for HP BASIC running on an HP Series 200 computer.

TMSL Overview

What is TMSL?

TMSL—the Test and Measurement Systems Language—is a programming language designed specifically for electronic test and measurement instruments. It defines how you communicate with these instruments from an external controller (computer).

Related Standards

Computer-controlled test instruments that were introduced in the 1960s used a wide variety of non-standard interfaces and communication protocols. During this time, Hewlett-Packard began developing the HP-IB as an internal standard. HP-IB defined a standard electrical and mechanical interface for connectors and cables. It also defined handshaking , addressing, and general protocol for transmitting individual bytes of data between instruments and computers.

IEEE 488.1

In 1975, the Institute of Electrical and Electronic Engineers (IEEE) approved IEEE 488-1975, which was based on Hewlett-Packard's internal HP-IB standard. This standard has been updated and is now IEEE 488.1-1987. Today, Hewlett-Packard uses the HP-IB name to indicate that a particular instrument or controller has capabilities that conform to the IEEE 488.1 standard.

Although it solved the problem of how to send bytes of data between instruments and computers, IEEE 488 did not specify the data bytes' meanings. Instrument manufacturers freely invented new commands as they developed new instruments. The format of data returned from instruments varied as well. By the early 1980s, work began on additional standards that specified how to interpret data sent via the 488 bus.

IEEE 488.2

In 1987, the IEEE approved IEEE 488.2-1987. This standard defined the roles of instruments and controllers in a measurement system connected by the 488 bus (HP-IB). In particular, IEEE 488.2 described how to send commands to instruments and how to send responses to controllers. Although it explicitly defined some frequently used commands, it still left the naming of most commands to instrument manufacturers. This made it possible for two similar instruments to conform to 488.2, yet have entirely different command sets.

TMSL

TMSL goes beyond 488.2 by defining a standard set of programming commands. For a given measurement function (such as frequency), TMSL defines the specific commands used to access that function via the 488 bus. If two analyzers both conform to the TMSL standard, for example, you would use the same command to set each analyzer's center frequency.

Standard commands provide two advantages:

- If you know how to control functions on one TMSL instrument, you know how to control the same functions on any TMSL instrument.
- Programs written for a particular TMSL instrument are easily adapted to work with a similar TMSL instrument.

Figure 1-1 shows you how TMSL builds on the 488 standards.

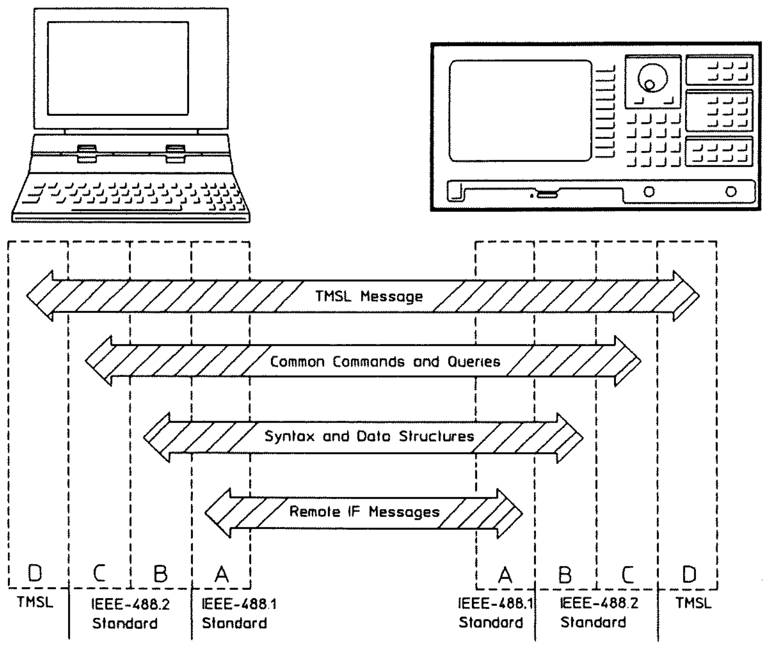


Figure 1-1. TMSL and Related Standards

The standards are layered to define different aspects of communication between devices:

- Layer A (IEEE 488.1) defines the physical and electrical connection between devices. It also defines how a byte of data is transmitted and how devices are instructed to talk and listen.
- Layer B (IEEE 488.2) defines the syntax and data formats used to send data between devices. It also defines the structure of status registers.
- Layer C (IEEE 488.2) defines the commands used for common tasks (such as resetting the device and reading the Status Byte).
- Layer D (TMSL) defines the commands used to control device-specific functions (such as setting frequency and amplitude). It also defines the parameters accepted by these functions and the values they return.

HP-IB Setup

This section contains a procedure for configuring the HP 3588A and an external controller in a simple HP-IB system. Although an HP 9836 computer is the controller used in the system, other computers that support an HP-IB interface can also be used. If you are using one of those other computers, the configuration procedure can only be used as a general guide. You should consult your computer's documentation for more complete information.

This section also contains a procedure for verifying that commands can be sent over the HP-IB. HP BASIC is used for the verification procedure's test program. If your computer uses some other language, the keywords and syntax for the test program may be different. You will need to write a similar program using your language's keywords and syntax.

Configuring the HP-IB System

Equipment and Software

- HP 3588A Spectrum Analyzer
- HP 9836 computer
- HP 10833A, B, C, or D HP-IB Cable
- HP BASIC

Procedure

1. Turn off the HP 3588A and the HP 9836, then connect them with the HP-IB cable as shown in figure 1-2.

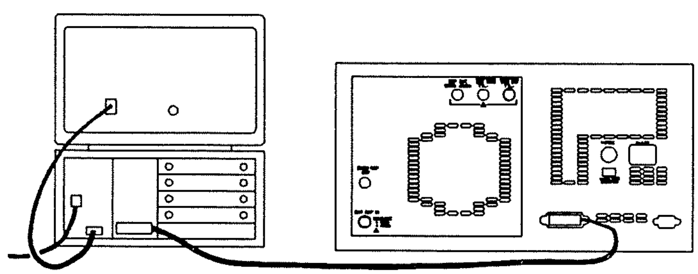


Figure 1-2. HP-IB Connections

2. Turn on the HP 9836. If necessary, load HP BASIC following the instructions in the computer's operating manual. Note that the following language extensions must be installed for the verification program to work:
   - CRTA.
   - HPIB.
   - IO.
   - EDIT.

   Programs that are more complex than the verification program will probably require more language extensions.

3. Turn on the HP 3588A. When the softkey labels appear, press the [**Local/HP-IB**] hardkey. (see figure 1-3)

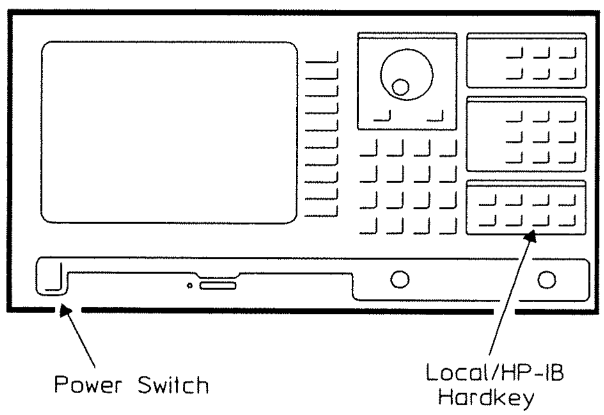


Figure 1-3. HP 3588A Front Panel

4. Verify that the analyzer's address is set to 19. The current address setting is displayed when you press the [ANALYZER ADDRESS] softkey (see figure 1-4). You can change the address by pressing [ANALYZER ADDRESS], then using the numeric keypad and the [ENTER] softkey to enter a new value. However, the instructions in the verification procedure assume that the analyzer address is set to 19.

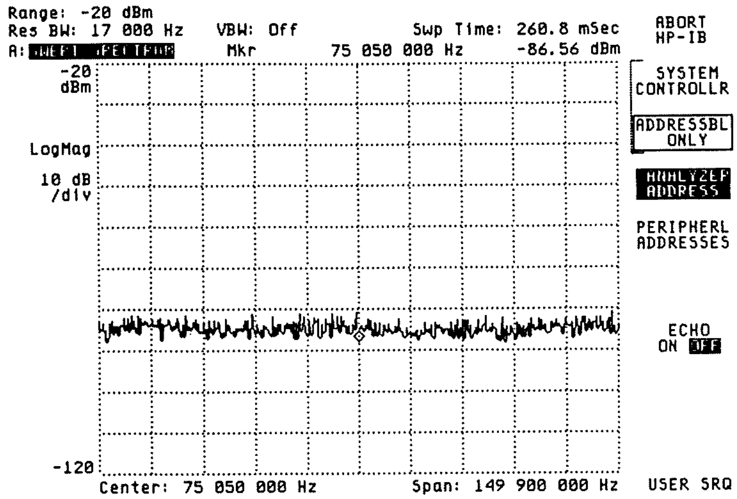


Figure 1-4. HP 3588A Screen After Pressing Local/HP-IB

5. Verify that the analyzer is set to the addressable-only mode. The softkey labels that appear when you press the [**Local/HP-IB**] hardkey include [SYSTEM CONTROLLR] and [ADDRESSBL ONLY]. Only one of these two softkeys can be selected at a time, and the one that is selected will have a box around it. If [ADDRESSBL ONLY] is not selected, then press that softkey.

Note

In any HP-IB system there can be more than one device with controller capabilities. But at any given time, only one device on the bus can be designated as the system controller.

Quick Verification

Having just completed all the steps in the preceding section, you are ready to verify that commands can be sent over the HP-IB. In this quick verification, you are going to enter an HP BASIC keyword that should place the HP 3588A under remote control.

Procedure

1. Type the following on the computer:

   ```
   REMOTE 719
   ```

   then press the computer's ENTER key. The RMT indicator should appear at the bottom of the HP 3588A's screen (see figure 1-5). This tells you that the analyzer is under remote control of the computer.

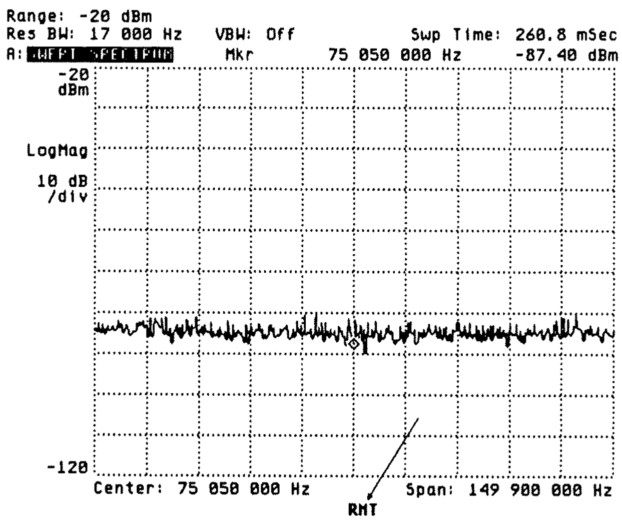


Figure 1-5. RMT Indicator

2. Now type the following on the computer:

   ```
   LOCAL 719
   ```

   then press the computer's ENTER key. The RMT indicator should disappear from the bottom of the screen. This tells you that the analyzer has been returned to front-panel control.

Troubleshooting

If the RMT indicator doesn't perform as expected, check the following things:

- Be sure that your HP-IB cable connections are secure and that the cable is free of defects.
- Verify that the analyzer is in addressable-only mode and that its address is set to 19.
- Be sure you are using the required equipment and software.
- Be sure you have loaded all the required language extensions into the computer. (For a list of loaded extensions, enter the following into the computer: LIST BIN)

If everything seems to be in order, but the RMT indicator still doesn't perform as expected, contact your local HP Sales/Service office.

Verification Program

The quick verification procedure confirmed that the computer could talk to the analyzer. However, you must write a short program to confirm that the analyzer can talk to the computer. If you enter the program correctly, the computer displays the following statement when you run the program:

FREQUENCY SPAN IS: +1.5E+8 HZ

Note The following procedure assumes that you have completed all the steps in "Configuring the HP-IB System" using all the required equipment and software.

Procedure

1. Enter the following program:

```
10      PRINTER IS 1
20      ASSIGN @Hp3588a TO 719
30      ABORT 7
40      CLEAR @Hp3588a
50      OUTPUT @Hp3588a;"*RST"
60      OUTPUT @Hp3588a;"SENS:FREQ:SPAN:FULL"
70      OUTPUT @Hp3588a;"SENS:FREQ:SPAN?"
80      ENTER @Hp3588a;A
90      PRINT "FREQUENCY SPAN IS:";A;"HZ"
100     END
```

2. See your computer and software documentation if you need help entering the program.

3. Press the computer's RUN key. The program tells the analyzer to preset. It then tells the analyzer to select its widest frequency span. Finally, the program asks the analyzer to return the value of the widest span and has the computer display the returned value as follows:

   FREQUENCY SPAN IS: +1.5E+8 HZ

Troubleshooting

If the program doesn't run correctly, be sure you have entered the program exactly as listed. Then go back to "Quick Verification" for additional troubleshooting hints.

Behavior in an HP-IB System

HP-IB Interface Capabilities

The HP 3588A has the following interface capabilities, as defined by the IEEE 488.1 standard:

SH1	complete Source handshake capability
AH1	complete Acceptor handshake capability
T6	basic Talker, Serial Poll, no Talk Only, unaddress if MLA
TE0	no Extended Talker capability
L4	basic Listener, no Listen Only, unaddress if MTA
LE0	no Extended Listener capability
SR1	complete Service Request capability
RL1	complete Remote/Local capability
PP0	no Parallel Poll capability
DC1	complete Device Clear capability
DT1	complete Device Trigger capability
C1	System Controller capability
C2	send IFC and take charge Controller capability
C3	send REN Controller capability
C12	send IF messages, receive control, pass control
E2	three-state drivers

Controller Capabilities

The HP 3588A can either be configured as an HP-IB system controller or as an addressable-only HP-IB device. This is done by selecting either the [SYSTEM CONTROLLR] or [ADDRESSBL ONLY] softkey on the analyzer's front panel. (These keys are displayed when you press the [**Local/HP-IB**] hardkey.)

Normally, the HP 3588A is not configured as the system controller unless it is the only controller on the bus. Such a setup would be likely if you just wanted to control printers or plotters with the analyzer. It might also be the case if you were using HP Instrument BASIC to control other test equipment.

When the analyzer is being used with another controller on the bus, it is normally configured as an addressable-only HP-IB device. In this configuration, the analyzer can function as the active controller (when it is passed control), or as a talker or listener.

Bus Management Commands vs. Device Commands

The HP-IB contains an attention (ATN) line that determines whether the interface is in command mode or data mode. When the interface is in command mode (ATN TRUE), a controller can send bus management commands over the bus. Bus management commands do the following things:

- Specify which devices on the interface can talk (send data) and which can listen (receive data).
- Instruct devices on the bus, either individually or collectively, to perform a particular interface operation.

The analyzer's responses to bus management commands are described in the next section.

When the interface is in data mode, device commands and data can be sent over the bus. Device commands are sent by the controller, but data can be sent either by the controller or a talker. The HP 3588A responds to two different kinds of device commands:

- *Common commands* access device functions required by the IEEE 488.2 standard.
- *Subsystem commands* access the bulk of the analyzer's functions.

The analyzer's responses to device commands are described in the Command Reference chapters.

Response to Bus Management Commands

This section tells you how the HP 3588A responds to the HP-IB bus management commands. The commands themselves are defined by the IEEE 488.1 standard. Refer to the documentation for your controller's language system to determine how to send these commands.

Device Clear (DCL)

The analyzer does the following things when it receives this command:

- Clears its input and output queues.
- Resets its command parser (so it is ready to receive a new program message).
- Cancels any pending *OPC command or query.

The command does not affect the following things:

- Front-panel operation.
- Any analyzer operations in progress (other than those already mentioned).
- Any analyzer settings or registers (although clearing the output queue may indirectly affect the Status Byte's MAV bit).

Go To Local (GTL)

This command returns the analyzer to local (front-panel) control. All keys on the analyzer's front-panel are enabled.

Group Execute Trigger (GET)

This command triggers the analyzer (causes it to start collecting measurement data) if the following two things are true:

- The trigger source is the HP-IB (TRIG:SOUR BUS).
- The analyzer is ready to trigger. (Bit 5 of the Standard Operation condition register is set.)

GET has the same effect as the *TRG and TRIG:IMM program messages with one important exception: *TRG and TRIG:IMM are sent to the input queue and processed in the order received, but GET is processed immediately, even if the input queue contains other commands.

Interface Clear (IFC)

This command causes the analyzer to halt all bus activity. It discontinues any input or output, although the input and output queues are not cleared. If the analyzer is designated as the active controller when this command is received, it relinquishes control of the bus to the system controller. If the analyzer is enabled to respond to a Serial Poll it becomes Serial Poll disabled.

Local Lockout (LLO)

This command causes the analyzer to enter the local lockout mode, regardless of whether it is in the local or remote mode. The analyzer only leaves the local lockout mode when the HP-IB's Remote Enable (REN) line is set FALSE.

Local lockout ensures that the analyzer's [**Local/HP-IB**] hardkey is disabled when the analyzer is in the remote mode. When the key is enabled, it allows a front-panel operator to return the analyzer to local mode, thus enabling all other front-panel keys. However, when the key is disabled, it does not allow the front-panel operator to return the analyzer to local mode.

Parallel Poll

The HP 3588A ignores all of the following parallel poll commands:

- Parallel Poll Configure (PPC).
- Parallel Poll Unconfigure (PPU).
- Parallel Poll Enable (PPE).
- Parallel Poll Disable (PPD).

Remote Enable (REN)

REN is a single line on the HP-IB. When it is set TRUE, the analyzer will enter the remote mode when addressed to listen. It will remain in remote mode until it receives the Go to Local (GTL) command or until the REN line is set FALSE.

When the analyzer is in remote mode and local lockout mode, all front-panel keys are disabled. When the analyzer is in remote mode but not in local lockout mode, all front-panel keys are disabled except for the [**Local/HP-IB**] hardkey. See Local Lockout for more information.

Selected Device Clear (SDC)

The analyzer responds to this command in the same way that it responds to the Device Clear command. See the latter for details.

Serial Poll

The analyzer responds to both of the serial poll commands. The Serial Poll Enable (SPE) command causes the analyzer to enter the serial poll mode. While the analyzer is in this mode, it sends the contents of its Status Byte register to the controller when addressed to talk.

When the Status Byte is returned in response to a serial poll, bit 6 acts as the Request Service (RQS) bit. If the bit is set, it will be cleared after the Status Byte is returned.

The Serial Poll Disable (SPD) command causes the analyzer to leave the serial poll mode.

Take Control Talker (TCT)

If the analyzer is addressed to talk, this command causes it to take control of the HP-IB. It becomes the active controller on the bus. The analyzer automatically passes control back when it completes the operation that required it to take control. Control is passed back to the address specified by the *PCB command (which should be sent prior to passing control).

If the analyzer does not require control when this command is received, it immediately passes control back.

Message Exchange

The analyzer communicates with the controller and other devices on the HP-IB via program messages and response messages. Program messages are used to send commands, queries, and data to the analyzer. Response messages are used to return data from the analyzer. The syntax for both kinds of messages is discussed in chapter 3.

There are two important things to remember about the message exchanges between the analyzer and other devices on the bus:

- The analyzer only talks after it receives a terminated query. (Query termination is discussed in "Query Response Generation," later in this chapter.)
- Once it receives a terminated query, the analyzer expects to talk before it is told to do something else.

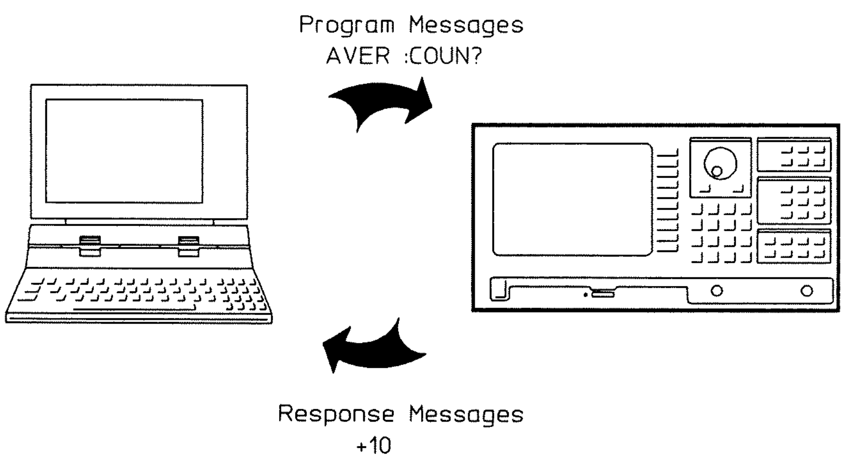


Figure 2-1. TMSL Message Exchange

HP-IB Queues

Queues enhance the exchange of messages between the HP 3588A and other devices on the bus. The analyzer contains:

- An input queue.
- An error queue.
- An output queue.

Input Queue

The input queue temporarily stores the following until they are read by the analyzer's command parser:

- Device commands and queries.
- The HP-IB END message (EOI asserted while the last data byte is on the bus).

The input queue makes it possible for a controller to send multiple program messages to the analyzer without regard to the amount of time required to parse and execute those messages. The queue holds up to 128 bytes. It is cleared when you do one of the following things:

- Turn on the analyzer.
- Send Device Clear (DCL) or Selected Device Clear (SDC).

Error Queue

The error queue temporarily stores up to 20 error messages. Each time the analyzer detects an error, it places a message in the queue. When you send the SYST:ERR query, one message is moved from the error queue to the output queue so it can be read by the controller. Error messages are delivered to the output queue in the order they were received. The error queue is cleared when you do one of the following things:

- Turn on the analyzer.
- Send the *CLS command.

Output Queue

The output queue temporarily stores a single response message until it is read by a controller. It is cleared when you do one of the following things:

- Turn on the analyzer.
- Send Device Clear (DCL) or Selected Device Clear (SDC).

Command Parser

The command parser reads program messages from the input queue in the order they were received from the bus. It analyzes the syntactic elements of the messages to determine what actions the analyzer should take.

One of the parser's most important functions is to determine the position of a program message in the analyzer's command tree. (For more information on the command tree, see chapter 3.) When the command parser is reset, the next syntactic element it receives is expected to arise from the base of the analyzer's command tree. The parser is reset when you do one of the following things:

- Turn on the analyzer.
- Send Device Clear (DCL) or Selected Device Clear (SDC).
- Follow a semicolon with a colon in a program message. (For more information, see "Sending Multiple Commands" in chapter 3.)

Query Response Generation

When the HP 3588A parses a query, the response to that query is placed in the analyzer's output queue. You should read a query response immediately after sending the query. This ensures that the response will not be cleared before it is read. The response will be cleared before you read it if either of the following message exchange conditions occur:

- Unterminated condition — This condition results when you neglect to properly terminate the query (with an ASCII line feed character or the HP-IB END message) before you read the response.

- Interrupted condition — This condition results when you send a second program message before reading the response to the first.

Synchronization

This section describes tools you can use to synchronize the analyzer and a controller. Proper use of these tools ensures that the analyzer will be in a known state when you send a particular command or query.

Overlapped Commands

Device commands can be divided into two broad classes:

- Sequential commands.
- Overlapped commands.

Most device commands that you send to the analyzer are processed sequentially. A sequential command holds off the processing of subsequent commands until it has been completely processed. However, some commands do not hold off the processing of subsequent commands; they are called overlapped commands.

Typically, overlapped commands take longer to process than sequential commands. For example, the SENS:REST command is used to restart a measurement. The command is not considered to have been completely processed until the measurement is complete. This can take a long time at narrow spans or when video averaging is enabled.

The analyzer uses a No Pending Operation (NPO) flag to keep track of overlapped commands that are still pending (not completed). The NPO flag is reset to 0 when an overlapped command is pending. It is set to 1 when no overlapped commands are pending. You cannot read the NPO flag directly, but all of the following common commands take some action based on the setting of the flag:

- *WAI — Holds off the processing of subsequent commands until the NPO flag is set to 1. Use this command to ensure that commands in the analyzer's input queue are processed in the order received.

- *OPC? — Places a 1 in the analyzer's output queue when the NPO flag is set to 1. Use this query to synchronize your controller to the completion of an overlapped command.

- *OPC — Sets bit 0 of the Standard Event event register to 1 when the NPO flag is set to 1. Use this command when you need to synchronize your controller to the completion of an overlapped command, but also want to leave the controller free to perform other tasks while the command is executing.

Each command requires a different amount of overhead in your program. *WAI requires the least overhead, *OPC requires the most.

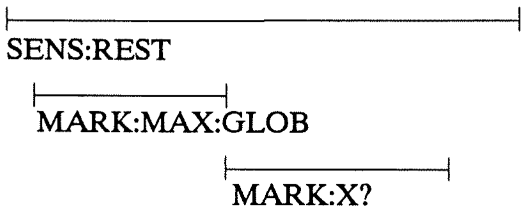
*WAI

This command holds off the processing of subsequent device commands until all overlapped commands are completed (the NPO flag is set to 1). The following example demonstrates the effect of the *WAI command.

Suppose you want to determine which frequency component of a signal contains the greatest amount of energy. You might send the following series of commands:

```
OUTPUT 719;"SENS:REST"              !Restart the measurement.
OUTPUT 719;"MARK:MAX:GLOB"          !Search for max energy.
OUTPUT 719;"MARK:X?"                             !Which frequency?
```

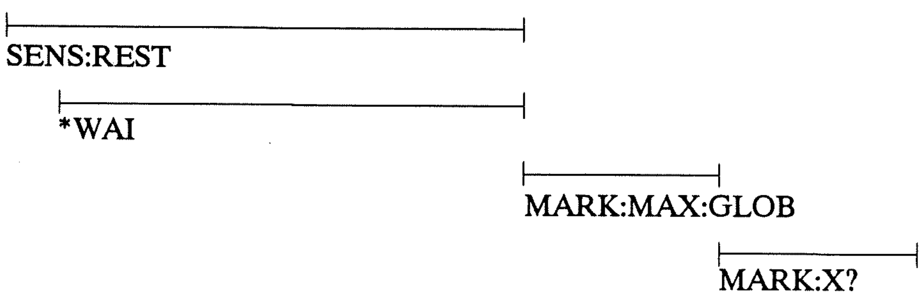
The following timeline shows how the processing times of the three commands relate to each other.



As you can see, SENS:REST is an overlapped command because it does not hold off the processing of MARK:MAX:GLOB. You may also recall that SENS:REST is not considered complete until the measurement is complete. So in this example, the marker searches for maximum energy before the measurement is complete. This may result in the MARK:X query returning an incorrect value. To solve the problem, you can insert a *WAI command.

```
OUTPUT 719;"SENS:REST"              !Restart the measurement.
OUTPUT 719;"*WAI"                                !Wait until complete.
OUTPUT 719;"MARK:MAX:GLOB"          !Search for max energy.
OUTPUT 719;"MARK:X?"                             !Which frequency?
```

The timeline now looks like this.



The *WAI command keeps the search from taking place until the measurement is completed. The MARK:X query will return the correct value.

*OPC? and *OPC

If you send *OPC?, a 1 is placed in the analyzer's output queue when the NPO flag is set to 1. This allows you to effectively pause the controller until all pending overlapped commands are completed. Just design your program so that it must read the queue before it continues.

If you send *OPC, bit 0 of the Standard Event register is set to 1 when the NPO flag is set to 1. This allows you to use the analyzer's register structure to generate a service request when all pending overlapped commands are completed. However, your program must also have enabled bit 0 of the Standard Event register and bit 5 of the Status Byte register. When you synchronize the analyzer and controller in this manner, the controller is free to perform some other task until the service request is generated.

*OPC does *not* hold off the processing of subsequent commands; it only informs you when the NPO flag is set to 1. As a result, you should not send any commands to the analyzer between the time you send *OPC and the time you receive a service request.

Passing Control

The HP 3588A requires temporary control of the HP-IB to complete some commands. (In the description of each command, a field called "Pass control required" indicates whether or not the command requires control of the bus.) After sending such a command, the active controller must pass control to the analyzer. When the analyzer completes the command, it automatically passes control of the bus back to the controller. For control to be passed back and forth smoothly, you must take steps to ensure that the following conditions are met:

- The analyzer must know the controller's address so it can pass control back.
- The controller must be informed when the analyzer passes control back.

Here is a procedure for passing control:

1. Send the controller's HP-IB address with the *PCB command.
2. Clear the analyzer's status registers with the *CLS command.
3. Enable the analyzer's status registers to generate a service request when the Operation Complete bit is set. (Send *ESE with a value of 1 and *SRE with a value of 32.)
4. Enable the controller to respond to the service request.
5. Send the command that requires control of the bus followed by the *OPC command.
6. Pass control to the analyzer and wait for the service request. The service request indicates that the command has been completed and control has been passed back to the controller.

Note

For this procedure to work properly, no overlapped commands should be pending except the command that requires control of the bus. For more information on overlapped commands, see "Synchronization" in this chapter.

---

chapter 6, "Programming Examples," contains an example program that passes control to the analyzer. In the example, control is passed so the analyzer can print the contents of its screen.

3

Programming with TMSL Commands

Introduction

This chapter will help you create more efficient programs with TMSL commands. It describes the general structure of the TMSL command tree and the syntax rules for TMSL program and response messages. It also explains how to do all of the following things:

- Send multiple commands.
- Shorten commands by abbreviating mnemonics.
- Shorten commands by omitting implied mnemonics.

The Command Tree

The TMSL standard organizes related instrument functions by grouping them together on a common branch of a command tree. Each branch is assigned a mnemonic to indicate the nature of the related functions. For example, the HP 3588A's marker functions are grouped together on the MARKER branch of the command tree. The MARKER branch is only one of 23 major TMSL branches—called subsystems—used by the HP 3588A.

When many functions are grouped together on a particular branch, additional branching is used to organize these functions into groups that are even more closely related. The MARKER branch serves as a good example because the analyzer provides many marker functions. Offset marker functions are grouped together on the OFFSET branch of the MARKER branch, peak search marker functions are grouped together on the MAXIMUM branch, and so on.

The branching process continues until each analyzer function is assigned to its own branch. For example, the function that turns the analyzer's peak track marker on and off is assigned to the TRACK branch of the MAXIMUM branch of the MARKER branch. The command looks like this:

```
MARKER:MAXIMUM:TRACK ON
```

Notice that branching points on the command tree are indicated with colons. Also notice that the parameter you assign to the function—ON in this case—is separated from the rest of the command by a space.

Sending Multiple Commands

You can send multiple commands within a single program message by separating the commands with semicolons. For example, the following program message—sent within an HP BASIC OUTPUT statement—would turn on the offset marker and move the main marker to the highest peak on the trace:

```
OUTPUT 719;"MARKER:OFFSET ON;:MARKER:MAXIMUM:GLOBAL"
```

The analyzer's command parser allows you to simplify the previous program message. This is because one of the parser's main functions is to keep track of a program message's position in the command tree. If you take advantage of this parser function, you can create the equivalent, but simpler, program message:

```
OUTPUT 719;"MARKER:OFFSET ON;MAXIMUM:GLOBAL"
```

In the first version of the program message, the semicolon that separates the two commands is followed by a colon. Whenever this occurs, the command parser is reset to the base of the command tree. As a result, the next command is only valid if it includes the entire mnemonic path from the base of the tree.

In the second version of the program message, the semicolon that separates the two commands is not followed by a colon. Whenever this occurs, the command parser assumes that the mnemonics of the second command arise from the same branch of the tree as the final mnemonic of the preceding command. OFFSET, the final mnemonic of the preceding command, arises from the MARKER branch. So MAXIMUM, the first mnemonic of the second command, is also assumed to arise from the MARKER branch.

Here is a longer series of commands—again, sent within HP BASIC OUTPUT statements—that can be combined into a single program message:

```
OUTPUT 719;"MARKER:STATE ON"
OUTPUT 719;"MARKER:OFFSET ON"
OUTPUT 719;"MARKER:MAXIMUM:GLOBAL"
OUTPUT 719;"MARKER:MAXIMUM:RIGHT"
```

The single program message would be:

```
OUTPUT 719;"MARKER:STATE ON;OFFSET ON;MAXIMUM:GLOBAL;RIGHT"
```

Command Abbreviation

Each command mnemonic has a long form and a short form. The short forms of the mnemonics allow you to send abbreviated commands. The mnemonics' short forms are created according to the following rules:

- If the long form of the mnemonic has less than four characters, the short form is the same as the long form. For example, ARM remains ARM.
- If the long form of the mnemonic has exactly four characters, the short form is the same as the long form. For example, USER remains USER.
- If the long form of mnemonic has more than four characters and the fourth character is a consonant, the short form consists of the first four characters of the long form. For example, AVERAGE becomes AVER.
- If the long form of mnemonic has more than four characters and the fourth character is a vowel, the short form consists of the first three characters of the long form. For example, LIMIT becomes LIM.

Note

The syntax descriptions in the Command Reference chapters use upper-case characters to identify the short form of a particular mnemonic.

If the rules listed in this section are applied to the last program message in the preceding section, the statement:

```
OUTPUT 719;"MARKER:STATE ON;OFFSET ON;MAXIMUM:GLOBAL;RIGHT"
```

becomes

```
OUTPUT 719;"MARK:STAT ON;OFFS ON;MAX:GLOB;RIGH"
```

Implied Mnemonics

You can omit some mnemonics from TMSL commands without changing effect of the command. These special mnemonics are called *implied mnemonics*, and they are used in many subsystems.

The MARKER subsystem contains the implied mnemonic STATE at its first branching point. As a result, you can send either of the following commands to the analyzer (using HP BASIC) to turn on the main markers:

```
OUTPUT 719;"MARKER:STATE ON"

OUTPUT 719;"MARKER ON"
```

The first mnemonic in the SENSE subsystem is also an implied mnemonic, so you can omit it from any SENSE command. These two commands are equivalent:

```
OUTPUT 719;"SENSE:FREQUENCY:SPAN:FULL"

OUTPUT 719;"FREQUENCY:SPAN:FULL"
```

and so are these:

```
OUTPUT 719;"SENSE:SWEEP:MODE AUTO"

OUTPUT 719;"SWEEP:MODE AUTO"
```

Message Syntax

As mentioned in chapter 2, the analyzer uses program messages and response messages to communicate with other devices on the HP-IB. This section uses syntax diagrams to describe the general syntax rules for both kinds of messages.

Conventions

The flow of syntax diagrams is generally from left to right. However, elements that repeat require a return path that goes from right to left. Any message that can be generated by following a diagram from its entry point to its exit point, in the direction indicated by the arrows, is valid.

Angle brackets < > enclose the names of syntactic items that need further definition. The definition is included either in the text accompanying the diagram, in a subsequent diagram, or in the next section, "Common Definitions."

The symbol ::= means "is defined as." When two items are separated by this symbol, the second item can replace the first in any statement that contains the first item.

Common Definitions

The syntax diagrams have the following definitions in common:

- <LF> is the line feed character (ASCII decimal 10).
- <^END> is assertion of the HP-IB END message while the last byte of data is on the bus.
- <SP> is the space character (ASCII decimal 32).
- <WSP> is one or more white space characters (ASCII decimal 0-9 and 11-32).
- <digit> is one character in the range 0-9 (ASCII decimal 48-57).
- <alpha> is one character of the alphabet. The character can be either upper-case (ASCII decimal 65-90) or lower-case (ASCII decimal 97-122) unless otherwise noted.

Special Syntactic Elements

Several syntactic elements have special meanings:

- colon (:) — When a command or query contains a series of mnemonics, the mnemonics are separated by colons. A colon immediately following a mnemonic tells the command parser that the program message is proceeding to the next level of the command tree. A colon immediately following a semicolon tells the command parser that the program message is returning to the base of the command tree. For more information, see "The Command Tree" and "Sending Multiple Commands" at the beginning of this chapter.

- semicolon (;) — When a program message contains more than one command or query, a semicolon is used to separate them from each other. For example, if you want to autorange the analyzer's inputs and then start a measurement using one program message, the message would be:

```
SENSE:POWER:RANGE:AUTO ONCE;:ABORT;:INITIATE:IMMEDIATE
```

- comma (,) — A comma separates the data sent with a command or returned with a response. For example, the SYSTEM:TIME command requires three values to set the analyzer's clock: one for hours, one for minutes, and one for seconds. A message to set the clock to 8:45 AM would be:

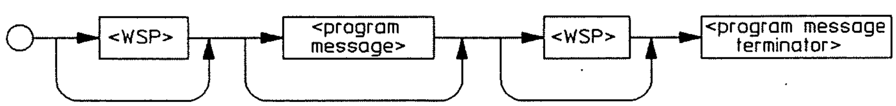
```
SYSTEM:TIME 8,45,0
```

- <WSP> — One or more white space characters are optional in many parts of a program message. However, at least one is required to separate a program header (the command or query) from its program data (the parameters). The previous example contains a space between the program header (SYSTEM:TIME) and its program data (8,45,0).

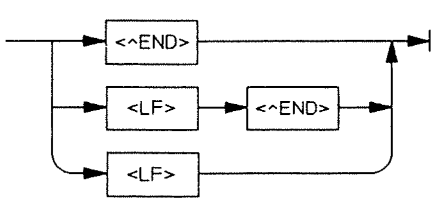
- <message terminator> — A message terminator is required at the end of a program message or a response message. Program message terminators are described in "Program Message Syntax." Response terminators are described in "Response Message Syntax."

Program Message Syntax

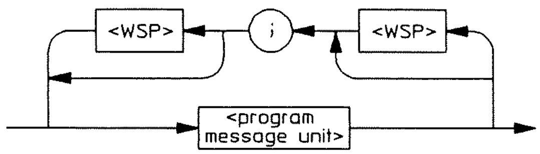
The syntax for a terminated program message is:



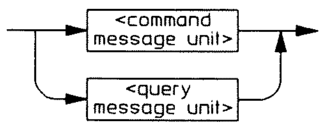
<program message terminator>::=



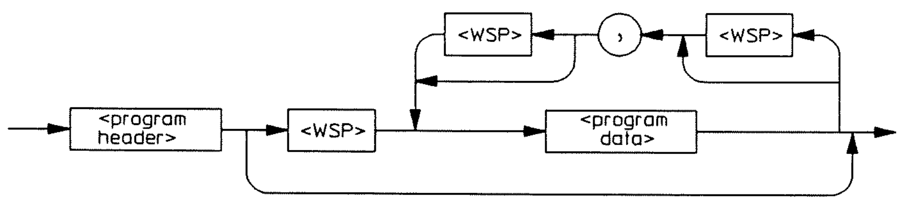
<program message>::=



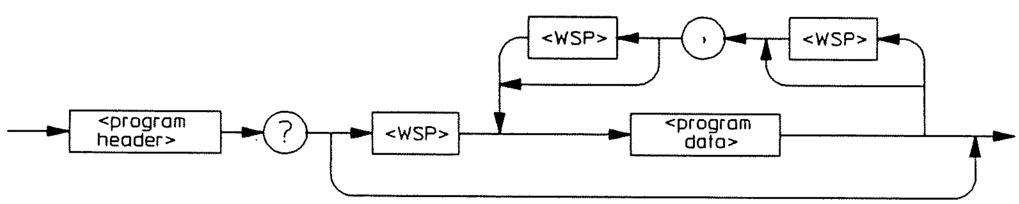
<program message unit>::=



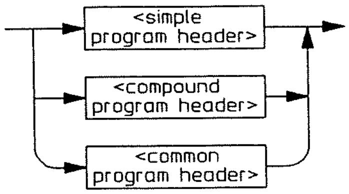
<command message unit>::=



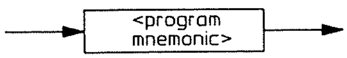
<query message unit>::=



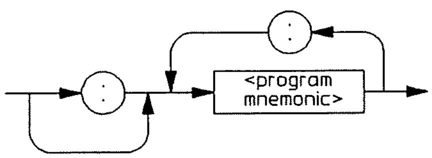
<program header>::=



<simple program header>::=

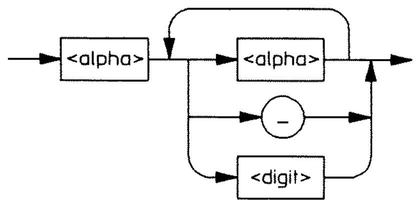


<compound program header>::=

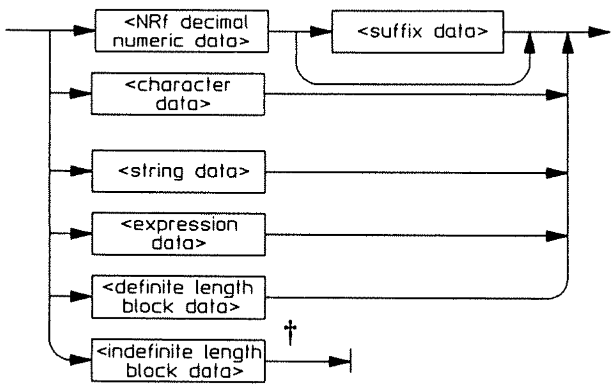


<common program header>::=

<program mnemonic>::=



<program data>::=



† The definition of indefinite length block data includes termination with <LF><^END>. This serves the dual function of terminating the data and terminating the program message.

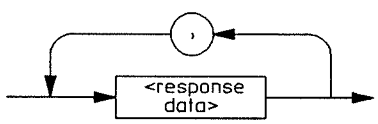
Program data and response data are described in chapter 4, "Transferring Data." <suffix data> is dependent on the command sent.

Response Message Syntax

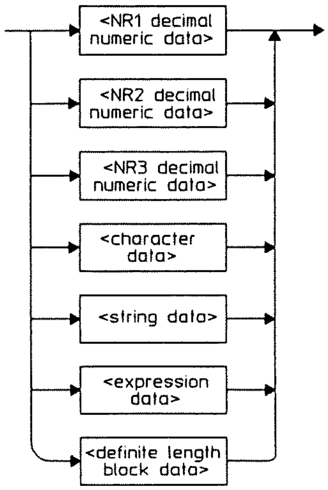
The syntax for a terminated response message is:

<response message terminator>::=

<response message>::=



<response data>::=



Response data and program data are described in chapter 4, "Transferring Data."

4

Transferring Data

Introduction

Data can be transferred between the HP 3588A and a controller via the HP-IB data lines, DIO1 through DIO8. Such transfers occur in a byte-serial (one byte at a time), bit-parallel (8 bits at a time) fashion. This chapter discusses the following aspects of data transfer:

- The different data types used during data transfers.
- Data encoding used during transfers of numeric block data.

Data Types

The HP 3588A uses a number of different data types during data transfers. They are described in this section using syntax diagrams.

Conventions

The flow of syntax diagrams is generally from left to right. However, elements that repeat require a return path that goes from right to left. Any data you can generate by following a diagram from its entry point to its exit point, in the direction indicated by the arrows, is valid.

Angle brackets < > enclose the names of syntactic items that need further definition. The definition is included either in the text accompanying the diagram, or in the next section, "Common Definitions."

Common Definitions

The syntax diagrams have the following definitions in common:

- <LF> is the line feed character (ASCII decimal 10).
- <^END> is assertion of the HP-IB END message while the last byte of data is on the bus.
- <SP> is the space character (ASCII decimal 32).
- <WSP> is one or more white space characters (ASCII decimal 0-9 and 11-32).
- <digit> is one character in the range 0-9 (ASCII decimal 48-57).
- <non-zero digit> is one character in the range 1-9 (ASCII decimal 49-57).
- <alpha> is one character of the alphabet. The character can be either upper-case (ASCII decimal 65-90) or lower-case (ASCII decimal 97-122) unless otherwise noted.

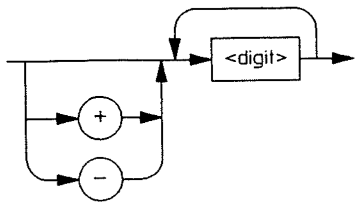
Decimal Numeric Data

The analyzer returns three types of decimal numeric data in response to queries:

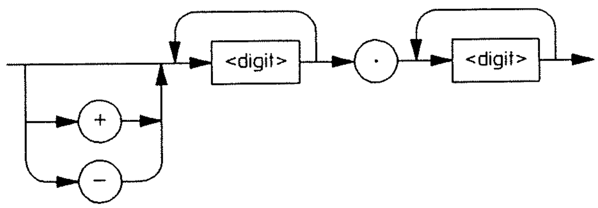
- Integers—returned as *NR1 data.*
- Fixed-point numbers—returned as *NR2 data.*
- Floating-point numbers—returned as *NR3 data.*

You can use the more flexible syntax of *NRf data* when sending any of the three decimal numeric data types to the analyzer. The NRx data syntax is described in the following four syntax diagrams.

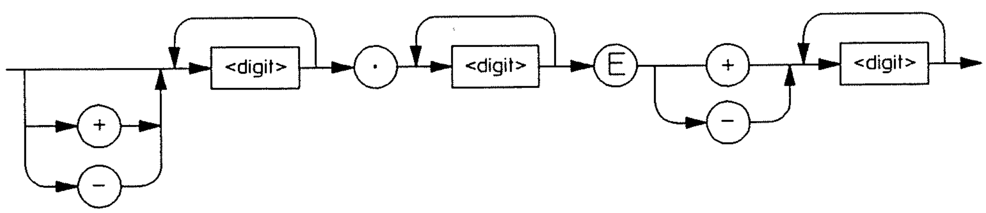
NR1 data:



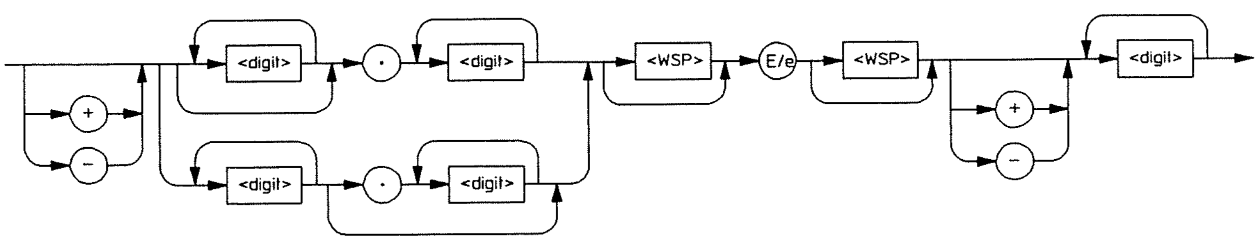
NR2 data:



NR3 data:

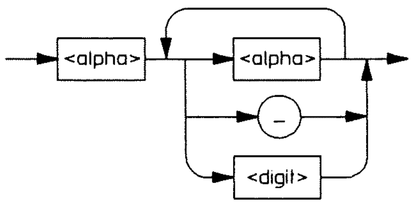


NRf data:



Character Data

The format you use to send character data is:

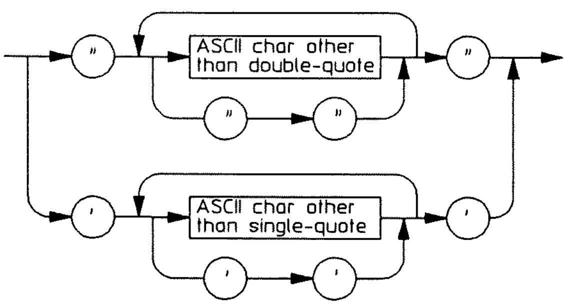


The "_" in the circle is the underscore character (ASCII decimal 95).

The format used when the analyzer returns character data is the same as the format used to send character data, with one exception—the analyzer never returns lower-case alpha characters.

String Data

The format you use to send string data is:

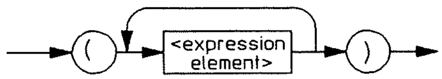


Note that you must use two double-quote characters ("") to represent one (") in a string that is delimited by double-quote characters. You must use two single-quote characters ('') to represent one (') in a string that is delimited by single-quote characters.

The format used when the analyzer returns string data is the same as the format used to send string data, with one exception: the analyzer never returns string data using the single-quote path.

Expression Data

The format you use to send expression data is:



The the only command that uses expression data is CALC:MATH:EXPR. The syntax description for that command contains a list of acceptable expression elements.

The analyzer returns expression data surrounded by double-quotes: "<expression data>".

Block Data

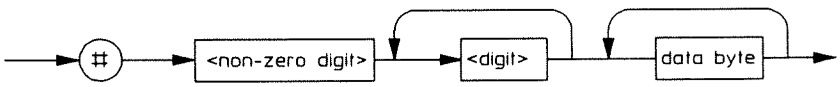
The analyzer returns one type of block data in response to queries: *definite length block data*. However, when you send block data to the analyzer, you can send it either as definite length or *indefinite length block data*.

Note

The analyzer always accepts block data, regardless of the setting of FORM:DATA. However, if FORM:DATA is set to ASC, values in the block must be encoded as 64-bit binary floating-point numbers. (For more information, see the next section of this chapter, "Data Encoding for Block Data."

Definite Length Block Data

The format you use to send definite length block data is:



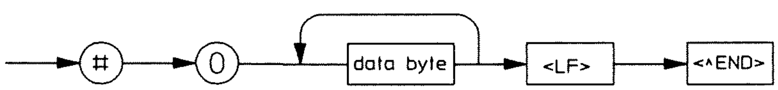
The elements #, <non-zero digit>, and <digit> make up a header for the block data. <non-zero digit> indicates how many times <digit> is repeated. The <digits> are interpreted as a single decimal number, which indicates how many bytes of data follow in the block. Here is an example:

Block Header				Block Data			
byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	...	byte 19
#	2	1	5	<data_byte_1>	<data_byte_2>	...	<data_byte_15>

<non-zero digit> is 2, which means that the following two bytes should be taken together as a single decimal number. In this case, the number is 15. The following 15 bytes are the 5th through 19th bytes of the data transfer, but they are the 1st through 15th bytes of the data block.

Indefinite Length Block Data

The format you use to send indefinite length block data is:



The first two bytes of the data transfer, # and 0, make up a header for the block data. The data itself does not begin until the third byte of the data transfer.

Data Encoding for Block Data

The FORM:DATA command selects the data type and data encoding that will be used to transfer large blocks of numeric data between the analyzer and an external controller:

- REAL selects the block data type (either the definite or indefinite length syntax). The block is transferred as a series of binary-encoded floating-point numbers.

- ASCii selects the decimal numeric data type (either the NR1, NR2, NR3, NRf syntax). The block is transferred as a series of ASCII-encoded NRx numbers separated by commas.

Blocks that contain mixed data—both numbers and ASCII characters—ignore the setting of FORM:DATA. They are always transferred as either definite or indefinite length block data. The following commands transfer blocks of mixed data:

- CALC:MATH:DATA
- DISP:LIM:LOW:DATA
- DISP:LIM:UPP:DATA
- PROG:SEL:DEF
- SYST:SET

ASCII Encoding

The ASCII 7-bit code is defined by the ANSI X3.4-1977 standard. When an ASCII-encoded byte is sent over the HP-IB, bits 0 through 6 of the byte (bit 0 being the least significant bit) correspond to the HP-IB data lines DIO1 through DIO7. DIO8 is ignored.

When you use ASCII encoding for block data, you can specify the number of significant digits to be returned for each number in the block. For example, if you send the command FORM:DATA ASC,7 then all numbers will be returned as NR3 data with 7 significant digits.

Binary Encoding

When you use binary encoding for block data, all numbers in the block are transferred as 32-bit or 64-bit binary floating-point numbers. The binary floating-point formats are defined in the IEEE 754-1985 standard. Send FORM:DATA REAL,32 to select the 32-bit format. Send FORM:DATA REAL,64 to select the 64-bit format.

Many controllers, and the languages that run on them, use the binary floating-point formats. Both formats have three fields in common, but the length of the fields are different for each. The fields and their bit lengths appear in the following table:

Table 4-1. Fields in Binary Floating-point Numbers

Field	Width of Field	
	32-bit format	64-bit format
sign (s)	1 bit	1 bit
exponent (e)	8	11
fraction (f)	23	52

When the 32-bit format is used, the decimal value of the exponent field ranges from −126 to +127, with a bias of +127. When the 64-bit format is used, the decimal value of the exponent field ranges from − 1022 to +1023, with a bias of +1023.

You can use the following formulas to determine the value (x) of a 32-bit binary floating-point number. (s, e, and f must be converted from binary to decimal before using the formulas.)

If $e = 255$ and $f \neq 0$	then x is not a number
If $e = 255$ and $f = 0$	then $x = -1^{s}(\infty)$
If $0 < e < 255$	then $x = -1^{s}(2^{e-127})(1 + f)$
If $e = 0$ and $f \neq 0$	then $x = -1^{s}(2^{e-126})(0 + f)$
If $e = 0$ and $f = 0$	then $x = -1^{s}(0)$

32-bit binary floating-point numbers are sent over the bus as follows:

DIO	8	7	6	5	4	3	2	1
byte 1	s	e	e	e	e	e	e	e
byte 2	e	f	f	f	f	f	f	f
bytes 3 and 4	f	f	f	f	f	f	f	f

You can use the following formulas to determine the value (x) of a 64-bit binary floating-point number. (Again, s, e, and f must be converted from binary to decimal before using the formulas.)

If $e = 2047$ and $f \neq 0$	then x is not a number
If $e = 2047$ and $f = 0$	then $x = -1^{s}(\infty)$
If $0 < e < 2047$	then $x = -1^{s}(2^{e-1023})(1 + f)$
If $e = 0$ and $f \neq 0$	then $x = -1^{s}(2^{e-1022})(0 + f)$
If $e = 0$ and $f = 0$	then $x = -1^{s}(0)$

64-bit binary floating-point numbers are sent over the bus as follows:

DIO	8	7	6	5	4	3	2	1
byte 1	s	e	e	e	e	e	e	e
byte 2	e	e	e	e	f	f	f	f
bytes 3 through 8	f	f	f	f	f	f	f	f

Here is an example of a number encoded in the 32-bit binary floating-point format:

byte 1	byte 2	byte 3	byte 4
01000001 seeeeeee	10010000 effffffff	00000000 ffffffff	00000000 ffffffff

Where:

	binary	decimal
s =	0	= 0
e =	10000011	= 131
f =	.001	= .125

Therefore:

$$
\begin{aligned}
x &= (-1)^{0}(2^{(131-127)})(1.125) \\
&= (2^{4})(1.125) \\
&= 18
\end{aligned}
$$

5

Using Status Registers

Introduction

The HP 3588A's status registers contain information about various analyzer conditions. This chapter describes the registers and tells you how to use them in your HP-IB programs. The registers are explained in the following sections:

- General Status Register Model
- The Service Request Process
- The HP 3588A's Register Sets

General Status Register Model

Overview

The general status register model, shown in figure 5-1, is the building block of the HP 3588A's status system. Most register sets in the analyzer include all of the registers shown in the general model (although commands are not always available for reading or writing a particular register). The information flow within a register set starts at the condition register and ends at the register summary bit. You control the flow by altering bits in the transition and enable registers.

Two register sets—Status Byte and Standard Event—are 8 bits wide. All others are 16 bits wide, but the most significant bit (bit 15) in the larger registers is always set to 0.

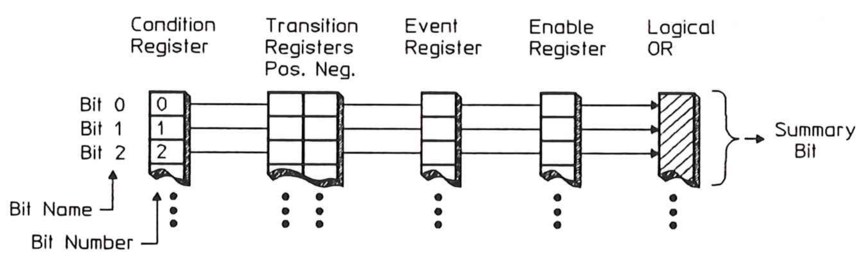


Figure 5-1. General Status Register Model

Condition Register

Condition registers continuously monitor hardware and firmware status. Bits in a condition register are not latched or buffered, they are updated in real time. When the condition monitored by a particular bit becomes true, the bit is set to 1. When the condition becomes false, the bit is reset to 0. Condition registers are read-only.

Transition Registers

Transition registers control the reporting of condition changes to the event registers. Positive changes in the state of a condition bit (0 to 1) are only reported to the event register if the corresponding positive transition bit is set to 1. Negative changes in the state of a condition bit (1 to 0) are only reported to the event register if the corresponding negative transition bit is set to 1. (If you set both transition bits to 1, positive *and* negative changes are reported to the corresponding event bit.) You can read and write most transition registers.

Event Register

Event registers latch any reported condition changes. When a transition bit allows a condition change to be reported, the corresponding event bit is set to 1. Once set, an event bit is no longer affected by condition changes. It remains set until the event register is cleared— either when you read the register or when you send the *CLS (clear status) command. Event registers are read-only.

An event register is cleared when you read it. All event registers are cleared when you send the *CLS command.

Enable Register

Enable registers control the reporting of events (latched conditions) to the register summary bit. If an enable bit is set to one, the corresponding event bit is included in the logical ORing process that determines the state of the summary bit. (The summary bit is only set to 1 if one or more enabled event bits are set to 1.) You can read and write all enable registers.

The Service Request Process

Two Ways to Use Registers

There are two methods you can use to access the information in status registers:

- The direct-read method.
- The service request (SRQ) method.

In the direct-read method, the analyzer has a passive role. It only tells the controller that conditions have changed when the controller asks the right question. In the SRQ method, the analyzer takes a more active role. It tells the controller when there has been a condition change without the controller asking. Either method allows you to monitor one or more conditions.

When you monitor a condition with the direct-read method, you must do the following things:

1. Determine which register contains the bit that monitors the condition.
2. Send the unique HP-IB query that reads that register.
3. Examine the bit to see if the condition has changed.

The direct-read method works well if you do not need to know about changes the moment they occur. It does not work well when you must know about condition changes immediately. Your program would need to continuously read the registers at very short intervals. Since this would make the program relatively inefficient, it would be better to use the SRQ method.

When you monitor a condition with the SRQ method, you must do the following things:

1. Determine which bit monitors the condition.
2. Determine how that bit reports to the request service (RQS) bit of the Status Byte.
3. Send HP-IB commands to enable the bit that monitors the condition and to enable the summary bits that report the condition to the RQS bit.
4. Enable the controller to respond to service requests.

When the condition changes, the analyzer sets its RQS bit and the HP-IB's SRQ line. Your program determines how the controller responds to the SRQ, but the important point is this: the controller is informed of the change as soon as it occurs. The time the controller would otherwise have used to monitor the condition can now be used to perform other tasks.

Generating a Service Request

To use the SRQ method, you must understand how service requests are generated. As shown in figure 5-2, other register sets in the HP 3588A report to the Status Byte. (Most of them report directly, but three report indirectly—via the Questionable Data register set.)

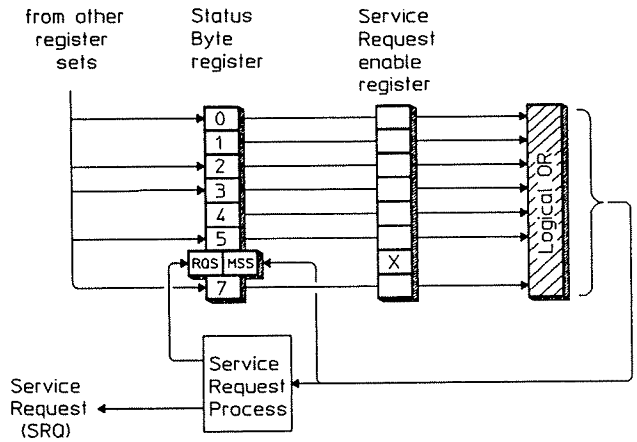


Figure 5-2. Generating a Service Request

When a register set causes its summary bit in the Status Byte to change from 0 to 1, the analyzer may initiate the service request (SRQ) process. However, the process is only initiated if both of the following conditions are true:

- The corresponding bit of the Service Request enable register is also set to 1.
- The analyzer does not have a service request pending. (A service request is considered to be pending between the time the analyzer's SRQ process is initiated and the time the controller reads the Status Byte register with a serial poll.)

The SRQ process sets the HP-IB's SRQ line true and also sets the Status Byte's request service (RQS) bit to 1. Both actions are necessary to inform the controller the HP 3588A requires service. Setting the SRQ line only informs the controller that *some* device on the bus requires service. Setting the RQS bit allows the controller to determine that the HP 3588A, in particular, requires service.

If your program enables the controller to detect and respond to service requests, it should instruct the controller to perform a serial poll when the HP-IB's SRQ line is set true. Each device on the bus returns the contents of its Status Byte register in response to this poll. The device whose RQS bit is set to 1 is the device that requested service.

Note

When you read the analyzer's Status Byte with a serial poll, the RQS bit is reset to 0. Other bits in the register are not affected.

As implied in figure 5-2, bit 6 of the Status Byte register serves two functions. Two different methods for reading the register allow you to access the two functions. Reading the register with a serial poll allows you to access the bit's RQS function. Reading the register with *STB allows you to access the bit's MSS function.

The HP 3588A's Register Sets

Register Summary

The HP 3588A uses nine register sets to keep track of instrument status:

- Status Byte.
- Device State.
- Limit Fail.
- Questionable Data.
- Questionable Frequency.
- Questionable Power.
- Standard Event.
- Standard Operation.
- User Defined.

Their reporting structure is summarized in figure 5-3. They are described in greater detail in the following sections.

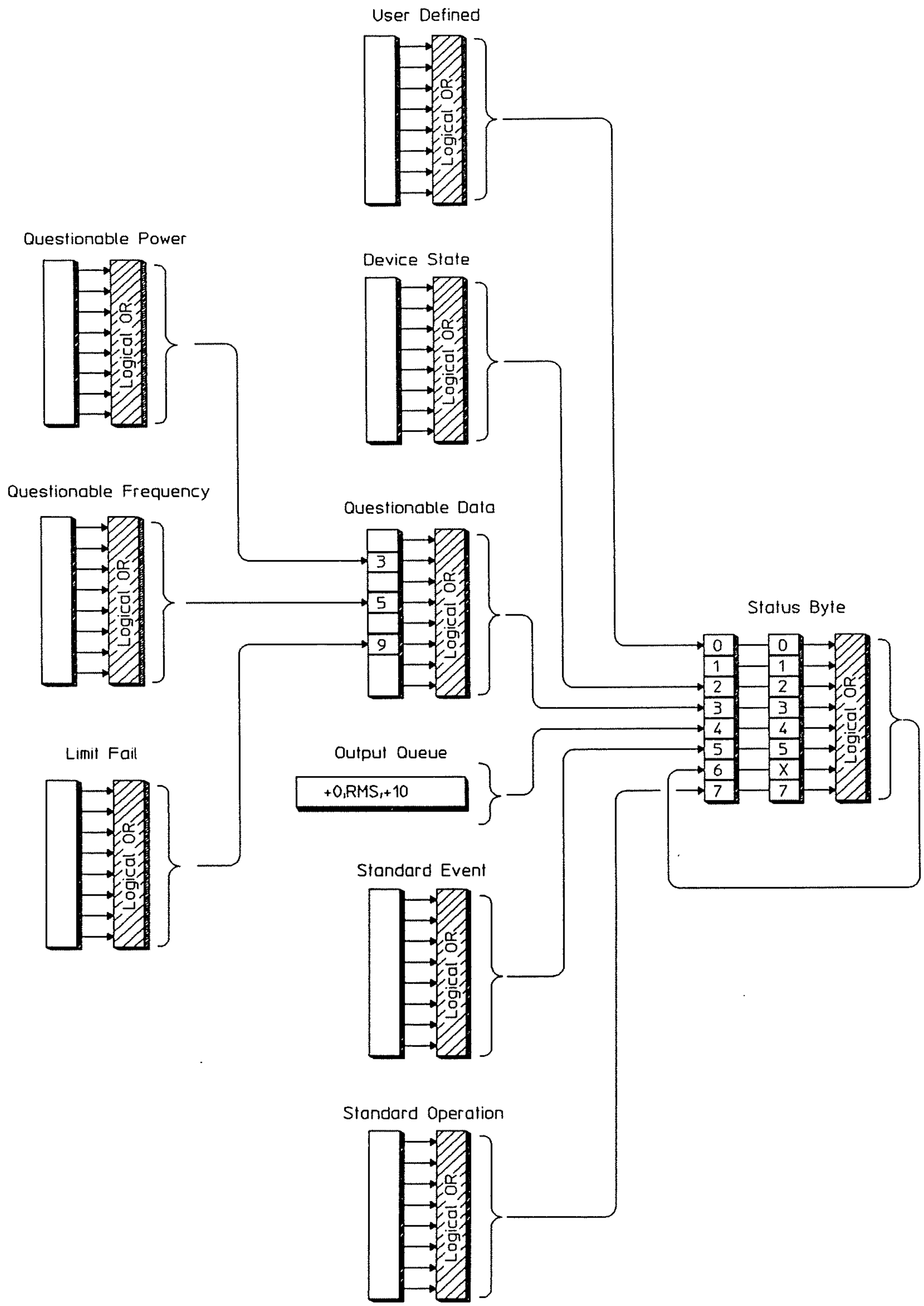


Figure 5-3. HP 3588A Register Summary

Status Byte Register Set

The Status Byte register set summarizes the states of the other register sets and monitors the analyzer's output queue. It is also responsible for generating service requests (see "Generating Service Requests" in this chapter).

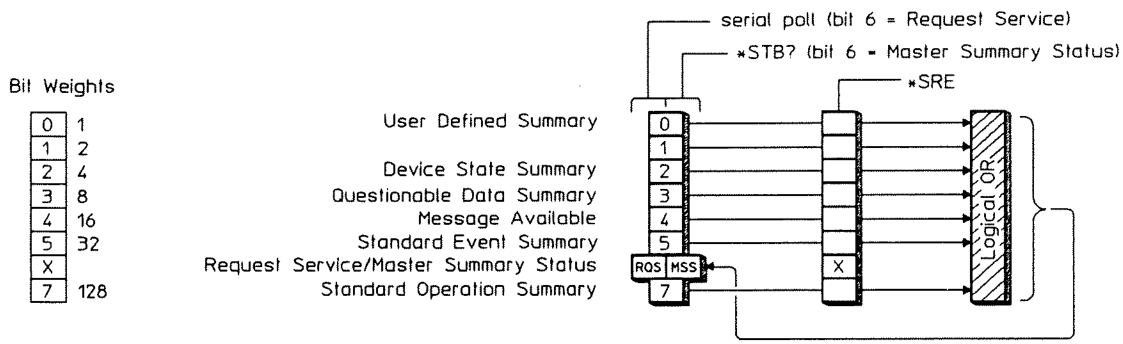


Figure 5-4. Status Byte Register Set

The Status Byte register set does not conform to the general status register model described at the beginning of this chapter. It contains only two registers: the Status Byte register and the Service Request enable register. The Status Byte register behaves like a condition register for all bits except bit 6. The Service Request enable register behaves like a standard enable register except that bit 6 is always set to 0.

Bits in the Status Byte register are set to 1 under the following conditions:

- User Defined Summary (bit 0) is set to 1 when one or more enabled bits in the User Defined event register are set to 1.
- Device State Summary (bit 2) is set to 1 when one or more enabled bits in the Device State event register are set to 1.
- Questionable Data Summary (bit 3) is set to 1 when one or more enabled bits in the Questionable Data event register are set to 1.
- Message Available (bit 4) is set to 1 when the output queue contains a response message.
- Standard Event Summary (bit 5) is set to 1 when one or more enabled bits in the Standard Event event register are set to 1.

- Master Summary Status (bit 6, when read by *STB) is set to 1 when one or more enabled bits in the Status Byte register are set to 1.

- Request Service (bit 6, when read by serial poll) is set to 1 by the service request process (see "Generating a Service Request" in this chapter).

- Standard Operation Summary (bit 7) is set to 1 when one or more enabled bits in the Standard Operation event register are set to 1.

Additional information is available under the commands you use to read and write the Status Byte registers. The commands are shown in figure 5-4; they are described in chapter 8.

Device State Register Set

The Device State register set monitors the states of three device specific parameters.

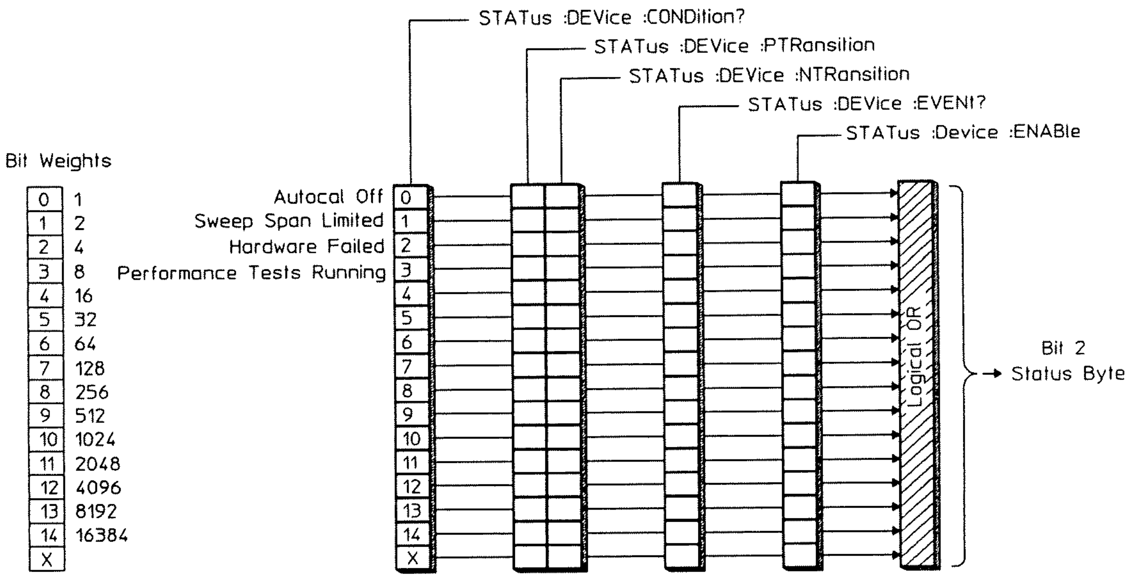


Figure 5-5. Device State Register Set

Bits in the Device State condition register are set to 1 under the following conditions:

- Autocal Off (bit 0) is set to 1 when the analyzer's autocalibration function is disabled (CAL:AUTO OFF).
- Sweep Span Limited (bit 1) is set to 1 when the current combination of span and center frequencies do not allow the measurement to fill an entire trace. (Trace values outside the range of 0 to 150 MHz are not allowed.)
- Hardware Failed (bit 2) is set to 1 when analyzer detects a failure in its own hardware.
- Performance Test Running (bit 3) is set to 1 when performance tests are running.

Additional information is available under the commands you use to read and write the Device State registers. The commands are shown in figure 5-5; they are described in chapter 27.

Limit Fail Register Set

The Limit Fail register set monitors limit test results for both traces.

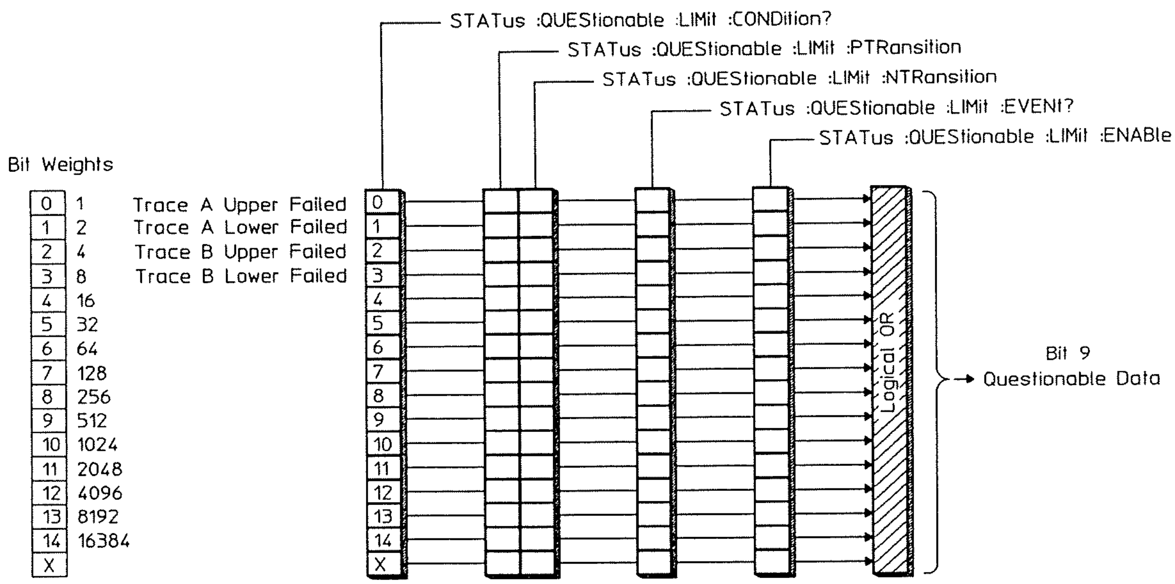


Figure 5-6. Limit Fail Register Set

Bits in the Limit Fail condition register are set to 1 under the following conditions:

- Trace A Upper Failed (bit 0) is set to 1 when any point on trace A exceeds its upper limit.
- Trace A Lower Failed (bit 1) is set to 1 when any point on trace A falls below its lower limit.
- Trace B Upper Failed (bit 2) is set to 1 when any point on trace B exceeds its upper limit.
- Trace B Lower Failed (bit 3) is set to 1 when any point on trace B falls below its lower limit.

Additional information is available under the commands you use to read and write the Limit Fail registers. The commands are shown in figure 5-6; they are described in chapter 27.

Questionable Data Register Set

The Questionable Data register set monitors conditions that affect the quality of measurement data.

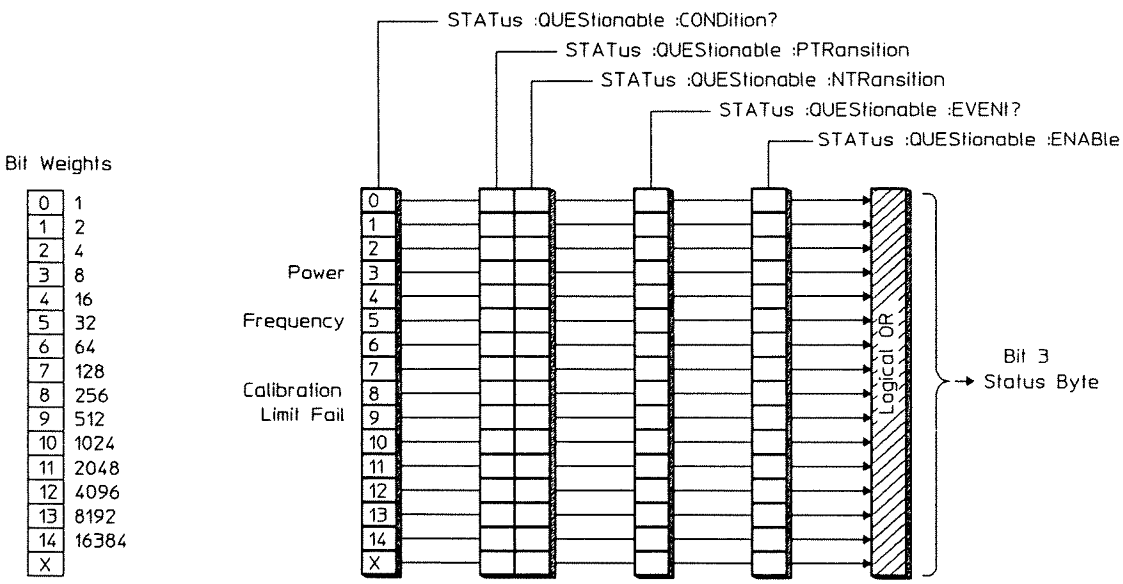


Figure 5-7. Questionable Data Register Set

Bits in the Questionable Data condition register are set to 1 under the following conditions:

- Power (bit 3) is set to 1 when one or more enabled bits in the Questionable Power event register are set to 1.
- Frequency (bit 5) is set to 1 when one or more enabled bits in the Questionable Frequency event register are set to 1.
- Calibration (bit 8) is set to 1 when the last self-calibration attempted by the analyzer failed.
- Limit Fail (bit 9) is set to 1 when one or more enabled bits in the Limit Fail event register are set to 1.

Additional information is available under the commands you use to read and write the Questionable Data registers. The commands are shown in figure 5-7; they are described in chapter 27.

Questionable Frequency Register Set

The Questionable Frequency register set monitors conditions that affect the frequency accuracy of measurement data.

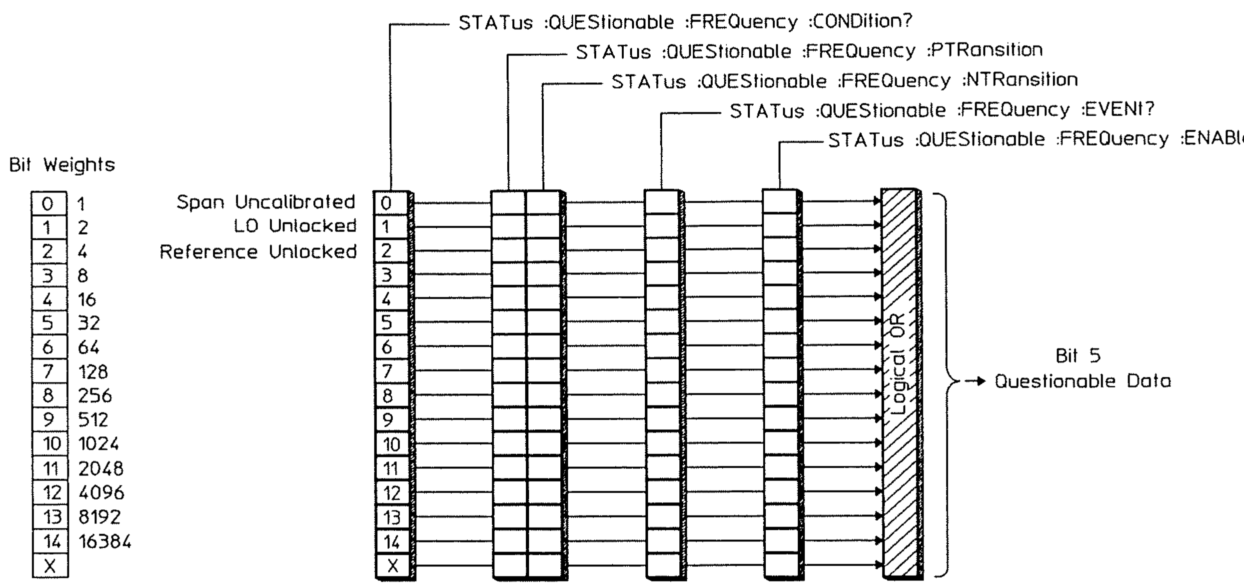


Figure 5-8. Questionable Frequency Register Set

Bits in the Questionable Frequency condition register are set to 1 under the following conditions:

- Span Uncalibrated (bit 0) is set to 1 when you request a span and sweep time that cannotbe realized.
- LO Unlocked (bit 1) is set to 1 when the analyzer's local oscillator is not locked to its internal reference signal(s).
- Reference Unlocked (bit 2) is set to 1 when the analyzer's internal reference signal is not locked to the external reference signal being applied to the analyzer's rear panel.

Additional information is available under the commands you use to read and write the Questionable Frequency registers. The commands are shown in figure 5-8; they are described in chapter 27.

Questionable Power Register Set

The Questionable Power register set monitors conditions that affect the amplitude accuracy of measurement data.

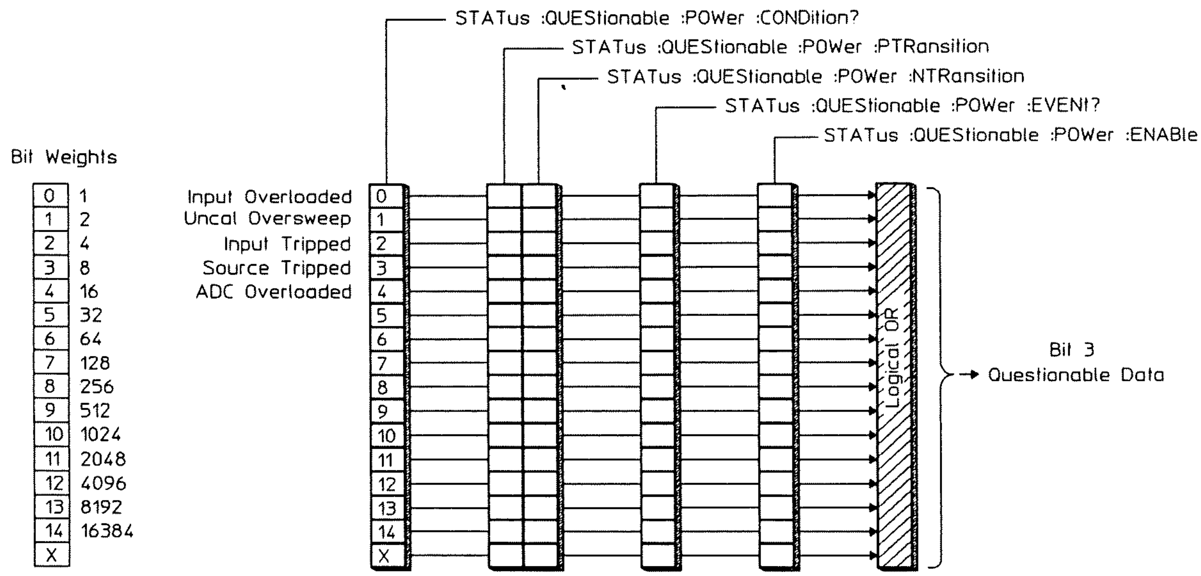


Figure 5-9. Questionable Power Register Set

Bits in the Questionable Power condition register are set to 1 under the following conditions:

- Input Overloaded (bit 0) is set to 1 when any signal between 0 and 150 MHz exceeds the current input range.
- Uncal Oversweep (bit 1) is set to 1 when the analyzer is sweeping too fast to provide accurate measurement results.
- Input Tripped (bit 2) is set to 1 when the analyzer's input-protection relay has been tripped (opened).
- Source Tripped (bit 3) is set to 1 when the analyzer's source-protection relay has been tripped (opened).
- ADC Overloaded (bit 4) is set to 1 when the analyzer's analog-to-digital converter is being overloaded.

Additional information is available under the commands you use to read and write the Questionable Power registers. The commands are shown in figure 5-9; they are described in chapter 27.

Standard Event Register Set

The Standard Event register set monitors TMSL errors and synchronization conditions.

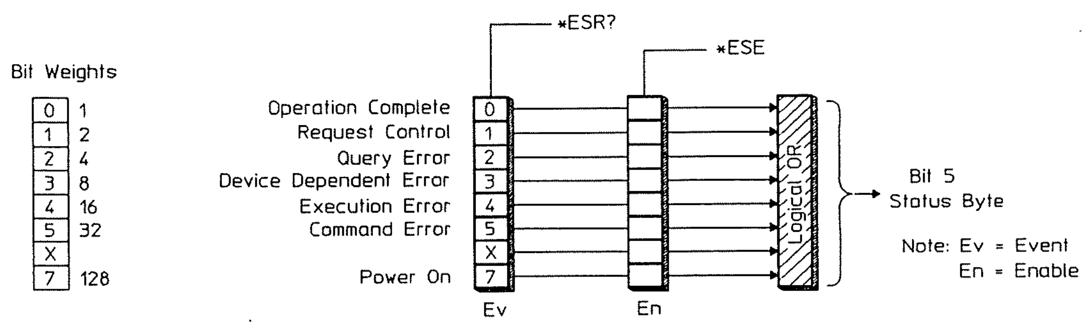


Figure 5-10. Standard Event Register Set

The Standard Event register set does not conform to the general status register model described at the beginning of this chapter. It contains only two registers: the Standard Event event register and the Standard Event enable register. The Standard Event event register is similar to other event registers, but it acts like conditions are always reported through a positive transition register that has all bits set to 1. The Standard Event enable register is the same as other enable registers.

Bits in the Standard Event event register are set to 1 under the following conditions:

- Operation Complete (bit 0) is set to one when the following two events occur (in the order listed):
  - You send the *OPC command to the analyzer.
  - The analyzer completes all pending overlapped commands (see "Synchronization" in Chapter 2).

- Request Control (bit 1) is set to 1 when both of the following conditions are true:
  - The analyzer is configured as an addressable-only HP-IB device (see "Controller Capabilities" in Chapter 2).
  - The analyzer is instructed to do something (such as plotting or printing) that requires it to take control of the bus.

- Query Error (bit 2) is set to 1 when the TMSL command parser detects a query error.
- Device Dependent Error (bit 3) is set to 1 when the TMSL command parser detects a device-dependent error.
- Execution Error (bit 4) is set to 1 when the TMSL command parser detects an execution error.
- Command Error (bit 5) is set to 1 when the TMSL command parser detects a command error.
- Power On (bit 7) is set to 1 when you turn the analyzer on.

Additional information is available under the commands you use to read and write the Standard Event registers. The commands are shown in figure 5-10; they are described in chapter 8.

Standard Operation Register Set

The Standard Operation register set monitors conditions in the analyzer's measurement process. It also monitors the state of the current HP Instrument BASIC program.

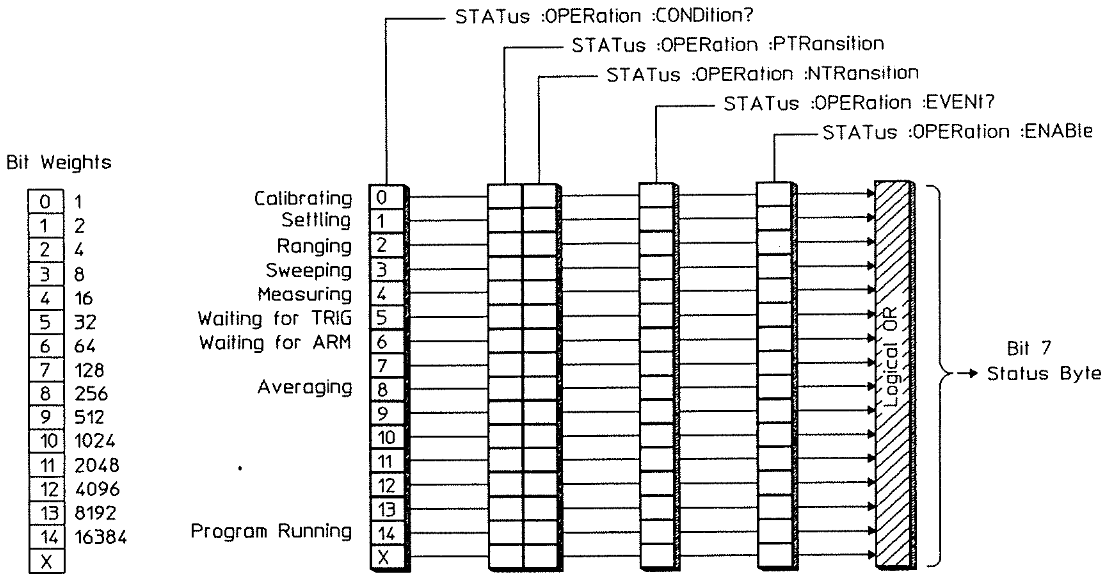


Figure 5-11. Standard Operation Register Set

Bits in the Standard Operation condition register are set to 1 under the following conditions:

- Calibrating (bit 0) is set to 1 while the self-calibration routine is running.
- Settling (bit 1) is set to 1 while the measurement hardware is settling.
- Ranging (bit 2) is set to 1 while the input range is changing.
- Sweeping (bit 3) is set to 1 during the data collecting portion of any swept spectrum measurement (including manual sweep and zero span).
- Measuring (bit 4) is set to 1 during the data collecting portion of any swept spectrum *or* narrow band zoom measurement.
- Waiting for TRIG (bit 5) is set to 1 when the analyzer is ready to accept a trigger signal from one of the trigger sources. (If a trigger signal is sent before this bit is set, the signal is ignored.)

- Waiting for ARM (bit 6) is set to 1 when both of the following conditions are true:
  - Manual arming is selected.
  - The analyzer is ready to be armed.

  (If you send the ARM:IMM command before this bit is set, the command is ignored.)

- Averaging (bit 8) is set to 1 when video averaging is enabled (AVER:TYPE VID *and* AVER:STAT ON).

- Program Running (bit 14) is set to 1 when the current HP Instrument BASIC program is running.

Additional information is available under the commands you use to read and write the Standard Operation registers. The commands are shown in figure 5-11; they are described in chapter 27.

User Defined Register Set

The User Defined register set detects STAT:USER:PULS commands and key-presses of the analyzer's [USER SRQ x] softkeys.

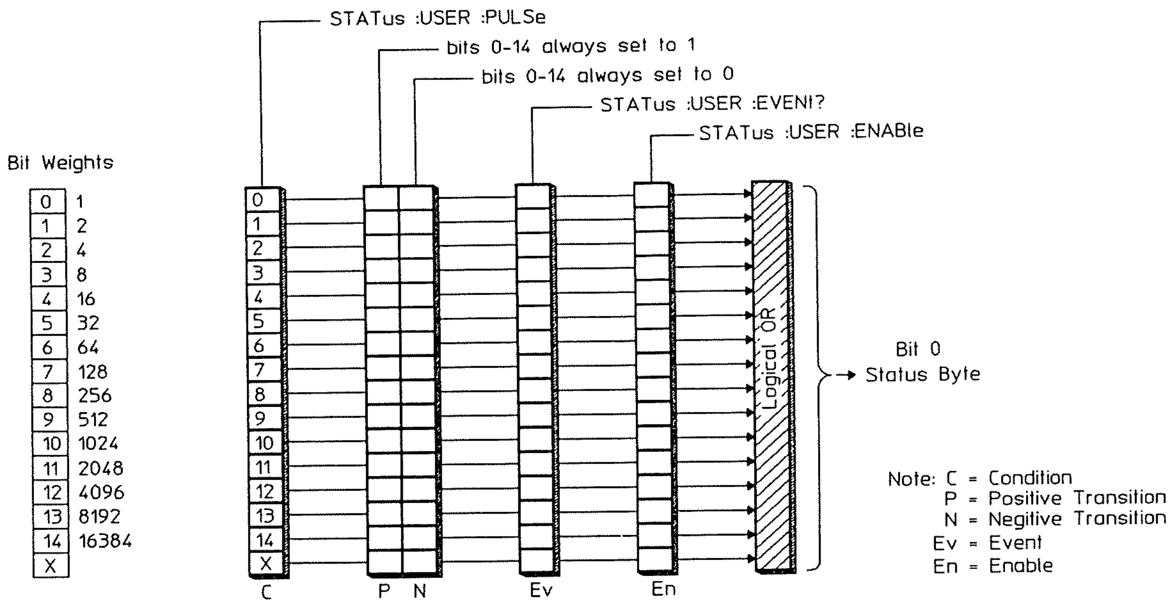


Figure 5-12. User Defined Register Set

The User Defined register set conforms to the general status register model (described at the beginning of this chapter) with the following exceptions:

- You can write (but not read) the condition register.
- You cannot write *or* read the transition registers.
- Bits in the positive transition register are always set to 1.
- Bits in the negative transition register are always set to 0.

Bits in the User Defined condition register are normally set to 0, but are set to 1 (briefly) when you press a [USER SRQ x] softkey or send a STAT:USER:PULS command. If you press [USER SRQ 5], bit 5 of the condition register is pulsed high. If you send STAT:USER:PULS 32, bit 5 of the condition register is pulsed high ($2^5 = 32$).

Programming Examples

This chapter contains listings of example programs written for the HP 3588A. All of the programs were written in HP BASIC for use on an HP Series 200/300 computer, although most are easily adaptable to other languages or programs. Most programs have been written to demonstrate one specific application and thus have been kept short and concise. The programs are grouped according to function as follows:

- Passing control
  - PASSCNTL
- Measurement synchronization
  - WAI_SYNC
  - OPC_SYNC
  - OPCQ_SYNC
- Generating SRQ's
  - RANGE_SRQ
  - AVER_SRQ
  - USER_SRQ
- Transferring data
  - MOVE_STATE
  - TRC_LOAD
  - LOG_XAXIS
- Measurement applications
  - SHAPE
  - THD
- Plotter applications
  - PLOT_CTRL

PASSCNTL

```
10    ! HP BASIC Program: PASSCNTL - Passing control to HP3588A
20    ! ---------------------------------------------------------------
30    ! This program instructs the HP 3588A perform a screen dump to a
40    ! printer and generate a service request when done.  Control is
50    ! passed to the HP 3588A when the print command is issued and
60    ! automatically passed back when the instrument no longer needs it.
70    !
80    Scode=7                            !Interface select code
90    Address=19                         !Address for HP3588A
100   Hp3588a=Scode*100+Address
110   OUTPUT Hp3588a;"*CLS"                  !Clear the STATUS BYTE register
120   !
130   ! Program the instrument to generate SRQ on OPERATION_COMPLETE. This
140   ! requires programming the STATUS BYTE and EVENT STATUS enable regs.
150   OUTPUT Hp3588a;"*ESE 1"           !Bit 1 = OPERATION_COMPLETE
160   OUTPUT Hp3588a;"*SRE 32"          !Bit 5 = EVENT_STATUS
170   OUTPUT Hp3588a;"*PCB 21"          !Set up Pass control back address
180   !
190   ON INTR Scode GOTO Srq_handler    !Set up interrupt branching
200   ENABLE INTR Scode;2               !Enable interrupt on SRQ
210   !
220   DISP "HP3588A Printing screen..."
230   OUTPUT Hp3588a;"PRIN:DUMP:ALL"    !Instruct analyzer to print the screen
240   OUTPUT Hp3588a;"*OPC"             !Set OPC bit when everythings complete
250   PASS CONTROL Hp3588a              !Give control of the bus to the 3588A
260   !
270 Wait_here:WAIT .5                   !Wait for OPC to generate an interrupt
280   GOTO Wait_here
290 !
300 Srq_handler:                        !If there's an interrupt, then
310                                     !Control was passed back
320   IF BINAND(SPOLL(Hp3588a),64) THEN !HP3588A is requesting service
330     BEEP
340     DISP "HP3588A Done Printing"
350   ELSE                              !It wasn't the HP3588A
360     DISP "UNKNOWN SRQ"
370   END IF
380   END
```

WAI_SYNC

```
10     ! HP BASIC program: WAI_SYNC -  Measurement synchronization
20     ! -------------------------------------------------------------------
30     ! This program demonstrates how to use the *WAI command to
40     ! prevent execution of an HP-IB command until all previous
50     ! commands have finished.  In this example, the trace display
60     ! will not change to the UPPER/LOWER FORMAT until after the
70     ! measurement has finished.
80     !
90     ! The *WAI command does not affect program operation. The
100    ! program will run to completion, sending all of the commands to
110    ! to the HP3588A without waiting for them to be executed.
120    ! -------------------------------------------------------------------
130    Scode=7                                !Interface select code
140    Address=19
150    Hp3588a=Scode*100+Address
160    !
161    DISP "Sending HP-IB commands..."
170    OUTPUT Hp3588a;"SCR:CONT TRACE;FORM SING" !Set format to single
180    OUTPUT Hp3588a;"SWE:TIME 8 S"          !Set record length to 8 seconds
190    OUTPUT Hp3588a;"ABORT; INIT"           !Start the measurement
200    OUTPUT Hp3588a;"*WAI"                  !Tell analyzer to wait here until
210                                           !all HP-IB commands have finished
220    OUTPUT Hp3588a;"SCR:CONT TRACE;FORM ULOW" !Go to upper/lower
230    BEEP
240    DISP "Finished. Display will go to UPPER/LOWER when meas. done"
250    END
```

OPC_SYNC

```
10    ! HP BASIC program: OPC_SYNC - Measurement synchronization
20    ! -------------------------------------------------------------------
30    ! This program demonstrates the how to use the *OPC command to
40    ! allow an SRQ to interrupt program execution.  *OPC will set
50    ! the OPERATION_COMPLETE bit in the EVENT STATUS register
60    ! when all pending HP-IB commands have finished.  With the proper
70    ! register masks, this will generate a service request.
90    !
100   Scode=7                             ! Interface select code
110   Address=19
120   Hp3588a=Scode*100+Address
130   !
140   OUTPUT Hp3588a;"SWE:TIME 8"         ! Set record length to 8 seconds
150   OUTPUT Hp3588a;"*CLS"               ! Clear the STATUS BYTE register
160   OUTPUT Hp3588a;"*ESE 1"             !Program the EVENT STATUS ENABLE reg.
170   OUTPUT Hp3588a;"*SRE 32"            !Program the STATUS BYTE ENABLE reg.
180   !
190   ON INTR Scode,2 GOTO Srq_handler    !Set up interrupt branching
200   ENABLE INTR Scode;2                 !Allow SRQ to generate an interrupt
210   !
220   OUTPUT Hp3588a;"ABORT; INIT"        !Start the measurement
230   OUTPUT Hp3588a;"*OPC"               !Generate SRQ when all commands have
240                                       !finished.
250   Start_time=TIMEDATE
260   LOOP                                !Do something useful while waiting
270     DISP USING "14A, 2D.D";"Elapsed time :",TIMEDATE-Start_time
280     WAIT .1
290   END LOOP
300   !
310 Srq_handler:    !Got an SRQ
320     Stb=SPOLL(Hp3588a)                    !Read STATUS BYTE and clear SRQ
330     BEEP
340     OUTPUT Hp3588a;"*ESR?"                !Read and clear EVENT STATUS reg.
350     ENTER Hp3588a;Esr
360     DISP "Got the SRQ! SPOLL returns:";Stb;"   ESR returns:";Esr
370   END
```

OPCQ_SYNC

```
10    ! HP BASIC program: OPCQ_SYNC -  Measurement synchronization
20    ! ------------------------------------------------------------------
30    ! This program demonstrates how to use the *OPC? HP-IB command
40    ! to suspend the bus prior to continuing on with the
50    ! program.  After all pending HP-IB commands have finished,
60    ! the HP 3588A will return a '1' in response to *OPC?.
70    !
80    Scode=7
90    Hp3588a=Scode*100+19
100   !
110   OUTPUT Hp3588a;"*RST"           !Preset the HP3588A
120   OUTPUT Hp3588a;"*OPC?"          !Pause on ENTER statement until
130   ENTER Hp3588a;Opc               !'*RST' command has finished
140   !
150   OUTPUT Hp3588a;"SWE:TIME 8"     !Set record length to 8 seconds
160   DISP "Measurement started ..."
170   OUTPUT Hp3588a;"ABOR; INIT"     !Start the measurement
180   OUTPUT Hp3588a;"*OPC?"          !Pause until all pending HP-IB commands
190   ENTER Hp3588a;Opc               !have finished.
200   BEEP
210   DISP "Measurement done"
220   END
```

RANGE_SRQ

```
10  ! HP BASIC program: RANGE_SRQ
20  ! ---------------------------------------------------------------------
30  !  This program demonstrates using the instrument's status registers
40  !  to enable SRQs for event initiated program interrupts.
50  !
60  ! ---------------------------------------------------------------------
70  !
80   Sc=7
90   Addr=19
100  Device=(Sc*100)+Addr
110  ASSIGN @Hp3588a TO Device
120  CLEAR SCREEN
130  !
140  ! Setup registers to detect range changes.
150  !
160  OUTPUT @Hp3588a;"*CLS"                ! CLEAR REGISTERS
170  OUTPUT @Hp3588a;"*SRE 128"            ! ENABLE OPERATIONAL STATUS SUMMARY
180  OUTPUT @Hp3588a;"STAT:OPER:ENAB 4"    ! ENABLE RANGE BIT
190  OUTPUT @Hp3588a;"STAT:OPER:PTR 4"     ! ENABLE POS TRANSITIONS
200  OUTPUT @Hp3588a;"STAT:OPER:NTR 0"     ! DISBALE NEG TRANSITIONS
210  !
220  ON INTR Sc GOSUB Check_srq
230  ENABLE INTR Sc;2
240  LOCAL 7
250  DISP " Press [Range/Input] hardkey then [SINGLE AUTORANGE] softkey"
260  ! Wait for SRQ
270  !
280 Hang_out:  GOTO Hang_out
290  !
300 Check_srq:    !
310  !
320    PRINT "SRQ Received"
330    Sb=SPOLL(Device)
340    PRINT "SPOLL(";Device;") = ";Sb
350    Send_query(@Hp3588a,"STAT:OPER:EVEN?")
360    PRINT
370    ENABLE INTR Sc
380    LOCAL 7
390    DISP " Press [Range/Input] hardkey then [SINGLE AUTORANGE] softkey"
400    RETURN
410  !
420  END
430  !**************************************************
440  ! Send a query command and print the return value.
450  !**************************************************
460  SUB Send_query(@Device,Cmd$)
470  OUTPUT @Device;Cmd$
480   ENTER @Device;Resp
490   PRINT Cmd$;" ";Resp
500   SUBEND
```

AVER_SRQ

```
! HP BASIC program: AVER_SRQ
! ------------------------------------------------
! This program demonstrates using the instrument's
! status registers and SRQs for sensing sweep
! completions while making a multi-sweep averaged
! measurement.
! ------------------------------------------------
!
Sc=7
Addr=19
Device=(Sc*100)+Addr
ASSIGN @Hp3588a TO Device
!
! Setup registers to detect measurement complete.
!
OUTPUT @Hp3588a;"*CLS"                ! CLEAR REGISTERS
OUTPUT @Hp3588a;"*SRE 128"            ! ENABLE OPERATIONAL STATUS SUMMARY
OUTPUT @Hp3588a;"STAT:OPER:ENAB 16"   ! ENABLE MEAS BIT
OUTPUT @Hp3588a;"STAT:OPER:PTR 0"     ! DISABLE POS TRANSITIONS
OUTPUT @Hp3588a;"STAT:OPER:NTR 16"    ! ENABLE NEG TRANSITIONS
!
ON INTR Sc GOSUB Check_srq
!
! Setup the instrument to take a 10 sweep average, each
! sweep individually armed.  HP-IB trigger (*TRG)
! could also be used.
!
OUTPUT @Hp3588a;"AVER:COUNT 10"
OUTPUT @Hp3588a;"AVER ON"
OUTPUT @Hp3588a;"ARM:SOUR MAN"
OUTPUT @Hp3588a;"ABORT;INIT"
FOR I=1 TO 10
   OUTPUT @Hp3588a;"ARM:IMM"
   IF I<10 THEN
        GOSUB Swp_wait
        DISP "Hit CONTINUE to take next sweep..."
        PAUSE
   END IF
NEXT I
OUTPUT @Hp3588a;"*OPC?"
ENTER @Hp3588a;X
DISP "ALL DONE!"
STOP
!
Swp_wait:     !
  DISP "Sweeping..."
  ENABLE INTR Sc;2
!
! Wait for SRQ
!
```

```
51 Hang_out:     !
52     IF NOT Sweep_done THEN
53          GOTO Hang_out
54     END IF
55       Sweep_done=0
56    RETURN
57    !
58 Check_srq:     !
59    !
60      Sb=SPOLL(Device)
61      Send_query(@Hp3588a,"STAT:OPER:EVEN?")
62      Sweep_done=1
63      RETURN
64    !
65    END
66    !**************************************************
67    ! Send a query command and print the return value.
68    !**************************************************
69    SUB Send_query(@Device,Cmd$)
70      OUTPUT @Device;Cmd$
71      ENTER @Device;Resp
72    SUBEND
```

USER_SRQ

```
10     ! HP BASIC Program: USER_SRQ - Responding to USER SRQ's
20     ! ----------------------------------------------------------------
30     ! Responding to User SRQ's
40     Hp3588a=719                        !Address of HP 3588A
50     INTEGER User_status_reg
60     !
70     OUTPUT Hp3588a;"*CLS"              !Clear all of the Status Registers
80     !
90     !Set USER STATUS REGISTER MASK to all 1's.
100    OUTPUT Hp3588a;"STAT:USER:ENAB 32767"
110    !
120    !Set STATUS BYTE Mask to generate SRQ on USER_EVENT_STATUS only
130    OUTPUT Hp3588a;"*SRE 1"
140    LOCAL Hp3588a                      !Put the instrument in LOCAL mode
150    !
160    ! Instrument is set up;  Enable interupts to detect an SRQ
170    ON INTR 7 GOSUB Srq_handler        !Set up interrupt branching
180    ENABLE INTR 7;2                    !Enable interupt on SRQ
190    !
200    CLEAR SCREEN                       !Clear the alpha screen
210  Wait:DISP "On the HP 3588A, Press [Local/HP-IB] [USER SRQ] [SRQx]"
220    GOTO Wait                          !Wait for SRQ to occur
230    !
240  Srq_handler:   !
250    IF BINAND(SPOLL(Hp3588a),64) THEN    !Bit 6 set, 3588 requires service
260      OUTPUT Hp3588a;"STAT:USER:EVEN?"   !Read USER STATUS REGISTER
270      ENTER Hp3588a;User_status_reg
280         !
290         !  Check all 16 bits in the User_status_reg
300         !  Note: Bits 10-15 can only be set via HP-IB
310      FOR Usrq_number=0 TO 15
320        IF BIT(User_status_reg,Usrq_number) THEN
330          SELECT Usrq_number
340          CASE 0
350            GOSUB Service_usrq0   !Gosub service routine for USER SRQ 0
360          CASE 1
370            GOSUB Service_usrq1   !Gosub service routine for USER SRQ 1
380          CASE 2 TO 15
390            GOSUB Service_usrqx   !Goto service routine for other USER SRQ's
400          END SELECT
410        END IF
420      NEXT Usrq_number
430      ENABLE INTR 7                       !re-enable interupts
440      LOCAL 7                             !put the instrument in local mode
450    ELSE
460      BEEP
470      DISP "UNKNOWN INTERRUPT"            !Don't know how to handle other
480      STOP                                !interrupts.
490    END IF
500    RETURN
```

```
510 Service_usrq0:      !Service routine to handle USER SRQ 0
520   PRINT "USER PRESSED SRQ 0"
530   RETURN
540 Service_usrq1:      !Service routine to handle USER SRQ 1
550   PRINT "USER PRESSED SRQ 1"
560   RETURN
570 Service_usrqx:      !Service routine to handle other USER SRQ's
580   PRINT "USER SRQ was between 2 and 15"
590   RETURN
600   END
```

MOVE_STATE

```
10   ! HP BASIC program: MOVE_STATE
20   ! ----------------------------------------------------------------------
30   ! Program that demonstrates how to download and upload state files
40   ! on the HP3588A.
50   !-----------------------------------------------------------------------
60   !
70   INTEGER Dig_cnt,State(1:2000)
80   DIM Resp$[200]
90   ASSIGN @Hp3588a TO 719;FORMAT ON
100  !
110  CLEAR SCREEN
120  CLEAR @Hp3588a
130  OUTPUT @Hp3588a;"FORM:DATA REAL"
140  ! Upload state data via HP-IB
150  !
160  OUTPUT @Hp3588a;"SYST:SET?"
170  !
180  ENTER @Hp3588a USING "%,A,D";Resp$,Dig_cnt
190  !
200  IF (Resp$<>"#") OR (Dig_cnt<=0) THEN
210    PRINT "Not correct block mode"
220    BEEP
230    CLEAR @Hp3588a
240    STOP
250  END IF
260  !
270  DISP "Uploading state"
280  ENTER @Hp3588a USING "%,"&VAL$(Dig_cnt)&"D";Num_bytes
290  PRINT "Number of bytes to upload ";Num_bytes
300  ASSIGN @Hp3588a;FORMAT OFF
305  REDIM State(1:Num_bytes/2)
310  ENTER @Hp3588a;State(*)
320  ASSIGN @Hp3588a;FORMAT ON
330  ENTER @Hp3588a;Resp$ ! READ TERMINATING LF
340  !
350  Reply$="Y"
360  INPUT "Download state to instrument?",Reply$
370  IF ( UPC$(Reply$[1,1]) = "Y" ) THEN GOTO Download
380  !
390  ! Save state into file compatible with SAVE/RECALL STATE
400  !
410  DISP "Creating state file"
420  MASS STORAGE IS ":INTERNAL,4,0"
430  ON ERROR GOTO Create_file
440  PURGE "STATE"
450  !
460 Create_file: OFF ERROR
470  CREATE BDAT "STATE",1,Num_bytes
480  ASSIGN @F TO "STATE"
490  DISP "Writing file"
```

```
500  OUTPUT @F;State(*)
510  ASSIGN @F TO *
520  DISP "File transfer complete."
530  STOP
540  !
550  ! Download state via HP-IB
560  !
570 Download:  !
580  Reply$="Y"
590  INPUT "Download state to instrument?",Reply$
600  IF ( UPC$(Reply$[1,1]) = "Y" ) THEN
610    DISP "Definite block transfer ";
620    OUTPUT @Hp3588a USING "#,K,D,"&VAL$(Dig_cnt)&"D";"SYST:SET
#",Dig_cnt,Num_bytes
630  ELSE
640    DISP "Indefinite block transfer ";
650    OUTPUT @Hp3588a;"SYST:SET #0";
660  END IF
670  !
680  ! Upload state data via HP-IB
690  !
700  ASSIGN @Hp3588a;FORMAT OFF
710  OUTPUT @Hp3588a;State(*)
720  ASSIGN @Hp3588a;FORMAT ON
730  OUTPUT @Hp3588a;CHR$(10) END
740  DISP "complete."
750  !
760  END
```

TRC_LOAD

```
! HP BASIC program: TRC_LOAD
! -----------------------------------------------------------
! Program to demonstrate dumping and uploading a trace on the
! HP 3588 with HP BASIC.
!
! -----------------------------------------------------------
!
DIM Dump_data(1:401),Load_data(1:401)
INTEGER Num_pts
!
ASSIGN @Hp3588a TO 719
!
CLEAR SCREEN
OUTPUT @Hp3588a;"FORM:DATA REAL"
DISP "Dumping trace..."
Dump_trace(@Hp3588a,Dump_data(*),Num_pts,"TRAC:DATA?")
OUTPUT @Hp3588a;"ARM:SOUR MAN"
!
! Flip-flop data points before uploading
!
DISP "Reversing data points..."
FOR I=1 TO Num_pts
  Load_data(Num_pts-I+1)=Dump_data(I)
NEXT I
!
DISP "Uploading trace..."
Upload_trace(@Hp3588a,Load_data(*),"TRAC:DATA")
DISP
OUTPUT @Hp3588a;"ARM:SOUR IMM"
!
END
!#########################################################################
!
SUB Dump_trace(@Hp3588a,REAL Trace_data(*),INTEGER Num_pts,Command$)
!------------------------------------------------------------------------
! MODULE DESCRIPTION:
!   This module dumps a trace of data from the instrument.
!
! INPUTS:   @Hp3588a   : Device to dump data from.
!           Command$   : HP-IB mnemonic used to prompt the instrument
!                        for the trace of data.
! OUTPUTS:  Trace_data : Array of data received from instrument.
!           Num_pts    : Number of points dumped into Trace_data.
!
!------------------------------------------------------------------------
 DIM A$[10]
 INTEGER Dig_cnt
 CLEAR @Hp3588a
 OUTPUT @Hp3588a;Command$
 ASSIGN @Hp3588a;FORMAT ON
```

```
51    Dig_cnt=-1
52    ENTER @Hp3588a USING "%,A,D";A$,Dig_cnt
53    IF (A$<>"#") OR (Dig_cnt<=0) THEN
54      PRINT "NOT CORRECT BLOCK MODE"
55      CLEAR @Hp3588a
56    ELSE
57      ENTER @Hp3588a USING "%,"&VAL$(Dig_cnt)&"D";Num_pts
58      IF (Num_pts MOD 8=0) THEN
59        Num_pts=Num_pts DIV 8
60        ASSIGN @Hp3588a;FORMAT OFF
61        ENTER @Hp3588a;Trace_data(*)
62        ASSIGN @Hp3588a;FORMAT ON
63        ENTER @Hp3588a;A$             ! Read CR/LF
64      ELSE
65        PRINT Data_read;" not float size (divisible by 8)"
66        CLEAR @Hp3588a
67      END IF
68    END IF
69   SUBEND
70   !##########################################################################
71   !
72   SUB Upload_trace(@Hp3588a,REAL Trace_data(*),Command$)
73   !-------------------------------------------------------------------------
74   ! MODULE DESCRIPTION:
75   !   This module uploads a trace of data.
76   !
77   ! INPUTS:    @Hp3588a   : Device to dump data to.
78   !            Command$   : HP-IB mnemonic used to load the trace of
79   !                         data into the instrument
80   ! OUTPUTS:   Trace_data : Array of data to load into instrument.
81   !
82   !-------------------------------------------------------------------------
83    CLEAR @Hp3588a
84    OUTPUT @Hp3588a;"FORM:DATA REAL"
85    OUTPUT @Hp3588a;Command$;" #0";
86    ASSIGN @Hp3588a;FORMAT OFF
87    OUTPUT @Hp3588a;Trace_data(*),END
88    ASSIGN @Hp3588a;FORMAT ON
89   SUBEND
```

LOG_XAXIS

```
! HP BASIC program: LOG_XAXIS
! ----------------------------------------------------------------
! Program to demonstrate dumping, re-scaling and plotting a trace of
! transform-coordinated data on the computer display.
!
! ----------------------------------------------------------------
!
COM REAL X_axis(1:401),Start_freq,Stop_freq
DIM Dump_data(1:401)
INTEGER Num_pts
!
ASSIGN @Hp3588a TO 719
GRAPHICS ON
GCLEAR
CLEAR SCREEN
!
LOOP
  DISP "Dumping trace..."
  Dump_trace(@Hp3588a,Dump_data(*),Num_pts,"CALC:DATA?")
  DISP Num_pts;" points dumped."
  !
  Log_xaxis(@Hp3588a,Num_pts)   ! Compute log x-axis
  !
  OUTPUT @Hp3588a;"ARM:SOUR MAN"
  DISP "Plotting trace..."
  Plot_trace(Dump_data(*),X_axis(*),Num_pts)
  OUTPUT @Hp3588a;"ARM:SOUR IMM"
  !
  DISP "Press CONTINUE for next trace"
  PAUSE
END LOOP
!
END
!######################################################################
!
SUB Dump_trace(@Hp3588a,REAL Trace_data(*),INTEGER Num_pts,Command$)
!---------------------------------------------------------------------
! MODULE DESCRIPTION:
!   This module dumps a trace of data from the instrument.
!
! INPUTS:    @Hp3558a    : Device selector of instrument.
!            Command$    : HP-IB mnemonic used to prompt the instrument
!                          for the trace of data.
! OUTPUTS:   Trace_data  : Array of data received from instrument.
!            Num_pts     : Number of points dumped into Trace_data.
!
!---------------------------------------------------------------------
 DIM A$[10]
 INTEGER Dig_cnt
 CLEAR @Hp3588a
```

```
51    OUTPUT @Hp3588a;"FORM:DATA REAL,64"
52    OUTPUT @Hp3588a;Command$
53    ASSIGN @Hp3588a;FORMAT ON
54    Dig_cnt=-1
55    ENTER @Hp3588a USING "%,A,D";A$,Dig_cnt
56    IF (A$<>"#") OR (Dig_cnt<=0) THEN
57      PRINT "NOT CORRECT BLOCK MODE"
58      CLEAR @Hp3588a
59    ELSE
60      ENTER @Hp3588a USING "%,"&VAL$(Dig_cnt)&"D";Num_pts
61      IF (Num_pts MOD 8=0) THEN
62        Num_pts=Num_pts DIV 8
63        ASSIGN @Hp3588a;FORMAT OFF
64        ENTER @Hp3588a;Trace_data(*)
65        ASSIGN @Hp3588a;FORMAT ON
66        ENTER @Hp3588a;A$              ! Read CR/LF
67      ELSE
68        PRINT Data_read;" not float size (divisible by 8)"
69        CLEAR @Hp3588a
70      END IF
71    END IF
72   SUBEND
73   !#########################################################################
74   !
75   SUB Plot_trace(REAL Trace_data(*),Xarray(*),INTEGER Num_pts)
76    !------------------------------------------------------------------------
77    ! MODULE DESCRIPTION:
78    !   This module plots a trace of data.
79    !
80    ! INPUTS:  Trace_data: Array of data to plot.
81    !          Xarray    : Array of x-axis values (corresponds to Trace_data)
82    !          Num_pts   : Number of points in Trace_data to plot.
83    !------------------------------------------------------------------------
84     REAL Y_scaler          ! 1/(Delta per vertical pixel)
85     REAL Max_val,Min_val   ! Maximum and minimum values in data trace
86     REAL Height            ! Display height
87     INTEGER Orig_x,Orig_y  ! Origin (in pixels)
88     !
89     Height=60
90     Orig_x=10
91     Orig_y=20
92     !
93     GCLEAR
94     Max_val=MAX(Trace_data(*))
95     Min_val=MIN(Trace_data(*))
96     !
97     ! Perform display auto-scale on data
98     !
99     Y_scaler=Height/(ABS(Max_val-Min_val))
100    X_scaler=.25           ! HP9836 Display
101    Top_pixel=(Height+Orig_y)
102    MOVE Orig_x,Top_pixel-(ABS(Max_val-Trace_data(1))*Y_scaler)
```

```
  FOR X=1 TO Num_pts
  Scaled_x = Orig_x+(Xarray(X)-1)*X_scaler
  Scaled_y = Top_pixel-(ABS(Max_val-Trace_data(X))*Y_scaler)
  DRAW Scaled_x, Scaled_y
  NEXT X
SUBEND
!
!######################################################################
!
SUB Log_xaxis(@Hp3588a,INTEGER Num_pts)
!--------------------------------------------------------------------
! MODULE DESCRIPTION:
!   This module computes a logarithmic x-axis for a linearly spaced
!   array of data.
!
! INPUTS:   @Hp3558a   : Device selector of instrument.
!           Num_pts    : Number of points in X_axis to calculate.
! OUTPUTS:  X_axis     : Array of x-axis values calculated.
!
!--------------------------------------------------------------------
COM REAL X_axis(1:401),Start_freq,Stop_freq
INTEGER I
REAL X,K,C,Inc,Temp_start,Temp_stop
!
  Displaywidth=401
!
  OUTPUT @Hp3588a;"FREQ:STAR?"
  ENTER @Hp3588a;Temp_start
  OUTPUT @Hp3588a;"FREQ:STOP?"
  ENTER @Hp3588a;Temp_stop
!
! Check for changes in frequency coverage
!
IF (Temp_start<>Start_freq) OR (Temp_stop<>Stop_freq) THEN
  Start_freq=Temp_start
  Stop_freq=Temp_stop
ELSE
  GOTO Bail
END IF
OUTPUT @Hp3588a;"ARM:SOUR MAN"
!
! Need to check for ~0 start frequency -> log crash
!
  DISP "Recalculating x-axis scaling"
!
  IF (Start_freq<.000000000001) THEN
    Start_freq=10.0
  END IF
  Inc=(Stop_freq-Start_freq)/(Num_pts-1)
  K=Displaywidth/(LGT(Stop_freq)-LGT(Start_freq))
  C=LGT(Start_freq)*(-K)
  I=1
  X=Start_freq
```

```
  LOOP
    EXIT IF ((X>Stop_freq) OR (I>Num_pts))
    X_axis(I)=C+K*LGT(X)
    I=I+1
    X=X+Inc
  END LOOP
!
OUTPUT @Hp3588a;"ARM:SOUR IMM"
Bail:SUBEND
```

SHAPE

```
1    ! HP BASIC program: SHAPE
2    ! ----------------------------------
3    ! Shape factor calculation test
4    ! ----------------------------------
5    !
6    REAL Meas_span,Peak_freq,Search_freq,Freq_per_bin
7    REAL Target_ampl,Left_freq,Right_freq
8    REAL Sixty_db_bw
9    INTEGER Slope_up
10   !
11 Shape_image: IMAGE "Shape factor = ",D.DD
12   !
13   ASSIGN @Hp3588a TO 719
14   !
15   ! Recall "3dB" State
16   !
17   OUTPUT @Hp3588a;"SYST:PRES"
18   !
19   Pass=1
20   LOOP
21     EXIT IF Pass>2
22     IF Pass=1 THEN
23       GOSUB Setup_3db
24     ELSE
25       GOSUB Setup_60db
26     END IF
27     DISP CHR$(129)&"Setting up filter measurement"&CHR$(128)
28     OUTPUT @Hp3588a;"REST; *WAI"     ! Take averaged measurement
29     OUTPUT @Hp3588a;"ARM:SOUR MAN"   ! Stop measurement
30     !
31     OUTPUT @Hp3588a;"DISP1:Y:SCAL:AUTO ONCE"
32     OUTPUT @Hp3588a;"FREQ:SPAN?"
33     ENTER @Hp3588a;Meas_span
34     Freq_per_bin=Meas_span/400.
35     OUTPUT @Hp3588a;"MARK:MAX:GLOB"
36     OUTPUT @Hp3588a;"MARK:FREQ?"
37     ENTER @Hp3588a;Peak_freq
38     !
39     OUTPUT @Hp3588a;"MARK:AMPL?"
40     ENTER @Hp3588a;Target_ampl
41     !
42     IF Pass=1 THEN
43       Target_ampl=Target_ampl-3.    ! (peak - 3 dB)
44       DISP CHR$(129)&"Calculating -3.0 dB Bandwidth"&CHR$(128)
45     ELSE
46       Target_ampl=Target_ampl-60.   ! (peak - 60 dB)
47       DISP CHR$(129)&"Calculating -60.0 dB Bandwidth"&CHR$(128)
48     END IF
49     !
50     ! Search for left target point
```

```
51    !
52    Right_freq=Peak_freq
53    Left_freq=Right_freq-(Meas_span/2.)
54    Slope_up=1
55    GOSUB Bin_search
56    OUTPUT @Hp3588a;"MARK:OFFS ON; OFFS:DELT:X 0; Y 0"
57    !
58    ! Search for right target point
59    !
60    Target_ampl=0.  ! Relative to OFFSET MARKER
61    Right_freq=Peak_freq+(Meas_span/2.)
62    Left_freq=Peak_freq
63    Slope_up=0
64    GOSUB Bin_search
65    !
66    OUTPUT @Hp3588a;"MARK:OFFS:DELT:X?"
67    IF Pass=1 THEN
68      ENTER @Hp3588a;Three_db_bw
69    ELSE
70      ENTER @Hp3588a;Sixty_db_bw
71    END IF
72    !
73    Pass=Pass+1
74    OUTPUT @Hp3588a;"MARK:OFFS OFF"
75  END LOOP
76  !
77  BEEP
78  DISP USING Shape_image;Sixty_db_bw/Three_db_bw
79  !
80  OUTPUT @Hp3588a;"ARM:SOUR IMM"  ! Re-start measurement
81  STOP
82  !
83 Bin_search:  !
84  !
85  ! Look for Target_ampl +- 0.01 dB
86  !
87  ! Left_freq and Right_freq bound area of search
88  !
89  Search_freq=Left_freq+(Meas_span/4.)
90  !
91  LOOP
92    OUTPUT @Hp3588a;"MARK:X ";Search_freq
93    IF Slope_up=0 THEN
94      OUTPUT @Hp3588a;"MARK:OFFS:DELT:Y?"  ! Searching for right target
95    ELSE
96      OUTPUT @Hp3588a;"MARK:Y?"
97    END IF
98    ENTER @Hp3588a;Search_ampl
99    EXIT IF (ABS(Target_ampl-Search_ampl)<=.01) OR
(ABS(Left_freq-Right_freq)<=Freq_per_bin)
100   !
101   IF Slope_up THEN
102      IF (Search_ampl>Target_ampl) THEN
```

```
103         Right_freq=Search_freq
104       ELSE
105         Left_freq=Search_freq
106       END IF
107    ELSE
108       IF (Search_ampl>Target_ampl) THEN
109         Left_freq=Search_freq
110       ELSE
111         Right_freq=Search_freq
112       END IF
113    END IF
114    !
115    Search_freq=Left_freq+(Right_freq-Left_freq)/2.
116   END LOOP
117   RETURN
118   !
119   ! Setup for 3 dB measurement
120 Setup_3db: !
121     OUTPUT @Hp3588a;"TEST:INP:CONF CAL"
122     OUTPUT @Hp3588a;"FREQ:CENT 10 MHz"
123     OUTPUT @Hp3588a;"BAND:AUTO OFF"
124     OUTPUT @Hp3588a;"FREQ:SPAN 2 kHz"
125     OUTPUT @Hp3588a;"BAND:RES 580 Hz"
126   RETURN
127   !
128   ! Setup for 60 dB measurement
129 Setup_60db: !
130     OUTPUT @Hp3588a;"ARM:SOUR IMM"   ! Re-start measurement
131     OUTPUT @Hp3588a;"FREQ:SPAN 5 kHz"
132   RETURN
133   END
```

THD

```
10    ! HP BASIC program: THD
20    ! -----------------------------------
30    ! Total Harmonic Distortion (THD) test
40    ! -----------------------------------
50    !
60    COM /Dut/ @Hp3588a
70    DIM Prompt$[60]
80    !
90 Fund_image:   IMAGE "    Fundamental : ",K," Hz, ", 3D.2D, " dBm"
100 Thd_image:  IMAGE  "    THD         : ", 2D.3D,"%  (", 3D.2D, " dB )"
110   !
120   ASSIGN @Hp3588a TO 719
130   !
140   Prompt$="Move marker past last harmonic, then press MEASURE THD."
150   !
160 Start:   !
170   !
180   OUTPUT @Hp3588a;"SYST:PRES"
190   !
200   CLEAR SCREEN
210   GCLEAR
220  !OUTPUT @Hp3588a;"TEST:INP:CONF CAL" ! THD of square wave cal signal
230   OUTPUT @Hp3588a;"SYST:RPGLOCK OFF"
240   DISP Prompt$
250   !
260   ON KEY 0 LABEL "MEASURE   THD" GOSUB Measure
270   FOR I = 1 TO 9
280     ON KEY I LABEL "" GOSUB Do_nothing
290   NEXT I
300 Hang_out: GOTO Hang_out
310   !
320 Do_nothing: RETURN
340 Measure:        !
350   CLEAR SCREEN
360   !
370   ! MEASURE LAST HARMONIC FREQUENCY LIMIT
380   !
390   OUTPUT @Hp3588a;"MARK:X?"
400   ENTER @Hp3588a;Last_x
410   !
420   ! MEASURE FUNDAMENTAL FREQUENCY
430   !
440   DISP "Frequency counting fundamental."
450   !
460   OUTPUT @Hp3588a;"ARM:SOUR IMM"
470   OUTPUT @Hp3588a;"MARK:MAX:GLOB"
480   OUTPUT @Hp3588a;"MARK:FUNC:FCO ON"
490   OUTPUT @Hp3588a;"REST; :ARM; *WAI"
500   OUTPUT @Hp3588a;"MARK:OFFS ON; OFFS:DELT:X 0; Y 0"
510   OUTPUT @Hp3588a;"MARK:X:FCO?"
```

```
520   ENTER @Hp3588a;Fund_x
530   OUTPUT @Hp3588a;"MARK:Y?"
540   ENTER @Hp3588a;Fund_y
550   OUTPUT @Hp3588a;"MARK:FUNC:FCO OFF"
560   PRINT
570   PRINT
580   PRINT USING Fund_image;Fund_x,Fund_y
590   !
600   DISP "Manually sweeping harmonics."
610   !
620   OUTPUT @Hp3588a;"SWE:MODE MAN"
630   !
640   Thd=0
650   Harm_x=2*Fund_x
660   !
670   LOOP
680     EXIT IF Harm_x>Last_x
690     OUTPUT @Hp3588a;"FREQ:MAN";Harm_x;"HZ"
700     OUTPUT @Hp3588a;"REST;  :ARM;  *WAI"
710     OUTPUT @Hp3588a;"MARK:X";Harm_x;"Hz"
720     OUTPUT @Hp3588a;"*WAI"
730     OUTPUT @Hp3588a;"MARK:OFFS:DELT:Y?"
740     ENTER @Hp3588a;Y
750     Thd=Thd+10.^(Y/10.)
760     Harm_x=Harm_x+Fund_x
770   END LOOP
780   !
790   Db=10.*LGT(Thd)
800   Per=100.*10.^(Db/20.)
810   PRINT
820   PRINT USING Thd_image;Per,Db
830   !
840   OUTPUT @Hp3588a;"SWE:MODE AUTO"
850   OUTPUT @Hp3588a;"ARM:SOUR IMM"
860   OUTPUT @Hp3588a;"MARK:OFFS OFF"
870   DISP Prompt$
880   RETURN
890   END
```

PLOT_CTRL

```
10    ! HP BASIC program: PLOT_CTRL
20    ! ------------------------------------------------------------------
30    !
40    ! Plot controller
50    !
60    ! Controls setting of rotation, P1 and P2 on HP-GL plotters for report
70    ! formatting.
80    !
90    ! For HP-GL plotters (at least 7475A) :
100   !
110   !   No rotation gives landscape format.
120   !   P1 specifies the lower left corner of the plot in (X,Y) coords
130   !   P2 specifies the upper right corner of the plot in (X,Y) coords
140   !
150   ! For 7475A :
160   !
170   !   1. 4 per page landscape :
180   !
190   !      * Rotation = 0 degrees
200   !      * Upper left quadrant   P1(1000,4500) P2(4700,7600)
210   !      * Upper right quadrant  P1(6000,4500) P2(9700,7600)
220   !      * Lower left quadrant   P1(1000,500)  P2(4700,3600)
230   !      * Lower right quadrant  P1(6000,500)  P2(9700,3600)
240   !
250   !   2. 2 per page portrait (side by side) :
260   !
270   !      * Rotation = 90 degrees
280   !      * Upper left  P1(100,7000) P2(4000,10000)
290   !      * Upper right P1(4500,7000) P2(7900,10000)
300   !
310   ! ------------------------------------------------------------------
320   !
330   INTEGER Rotate,P1_x,P1_y,P2_x,P2_y,Plot_done
340   ASSIGN @Hp3588a TO 719
345   OUTPUT @Hp3588a;"*PCB 21"
350   !
360   LOOP
370     DISP "Setup next screen to plot, then press CONTINUE."
380     PAUSE
390     !
400     ! Prompt user for rotation of plot.
410     !
420     INPUT "Enter rotation (degrees) : ",Rotate
430     OUTPUT 705;"RO ";Rotate;";" ! Send rotation to plotter
440     !
450     ! Prompt user for P1 and P2 for plot.
460     !
470     INPUT "Enter P1X, P1Y, P2X, P2Y : ",P1_x,P1_y,P2_x,P2_y
480     OUTPUT 705;"IP ";P1_x,P1_y,P2_x,P2_y;";"
490     !
```

```
500    OUTPUT @Hp3588a;"PLOT:DUMP:ALL"  ! Plot screen
510    CALL Passcntrl
520  END LOOP
530  !
540  END
550  !
560  !-----------------------------------------
570  ! Passes control to the analyzer and waits
580  ! for control to to be passed back.
590  !-----------------------------------------
600  SUB Passcntrl
610    PASS CONTROL 719
620    ON ERROR GOTO Not_done
630 Not_done:  !
640    WAIT 1
650    CLEAR 7
660    OFF ERROR
670  SUBEND
```

7

Introduction to the Command Reference

The command reference chapters describe all of the HP 3588A's TMSL commands. The command descriptions have the following things in common:

- A brief description of the command. This one- or two-line description appears just below the heading.
- A syntax description. This consists of one or two fields, depending on whether the command has just a command form, just a query form, or both. It shows you the syntax expected by the analyzer's TMSL command parser.
- Example statements. This field appears at the end of the first syntax description. It contains two HP BASIC output statements that use the command.
- A returned format description. This field is only used if the command has a query form. It tells you how data is returned in response to the query.
- An attribute summary. This field defines the command's preset state. It also defines attributes that affect command synchronization. (See "Synchronization" in chapter 2 for more information.)
- A detailed description. This field contains information that will help you use the command more efficiently.

Finding the Right Command

If you are looking for a command you've seen in a program, remember that commands can omit implied mnemonics. For example, the command SENSe:FREQuency:CENTer 10 MHZ contains the implied mnemonic SENSe. But SENSe can be omitted to create the equivalent command FREQuency:CENTer 10 MHZ. (See "Implied Mnemonics" in chapter 2.)

You will not find an entry for FREQuency:CENTer—or any other command that omits an implied mnemonic—in the command reference. If you don't find a command where you expect it, try scanning the command list in appendix A for the equivalent command that contains the implied mnemonic. After you locate the equivalent command, you can find its description in one of the command reference chapters.

If you are looking for a command that accesses a particular function, use the index. For example, if you want to find the command that changes the analyzer's center frequency, look for "center frequency" in the index. It sends you to the page that describes the SENSe:FREQuency:CENTer command.

Conventions

Syntax and returned format descriptions use the following conventions:

- < > Angle brackets enclose the names of syntactic items that need further definition. The definition will be included in accompanying text.

- ::= "is defined as" When two items are separated by this symbol, the second item can replace the first in any statement that contains the first item. For example, A::=B indicates that B can replace A in any statement that contains A.

- | "or" When items in a list are separated by this symbol, one and only one of the items can be chosen from the list. For example, A|B indicates that A or B can be chosen, but not both.

- ... An ellipsis (trailing dots) is used to indicate that the preceding element may be repeated one or more times.

- [ ] Square brackets indicate that the enclosed items are optional.

- { } Braces are used to group items into a single syntactic element. They are most often used to enclose lists and to enclose elements that are followed by an ellipsis.

In addition, the case of letters in the command mnemonics is significant. Mnemonics that are longer than four characters can have a short form or a long form. The analyzer accepts either form. Upper-case letters show the short form of a command mnemonic. For more information, see "Command Abbreviation" in chapter 3.

8

Common Commands

This chapter contains all of the IEEE 488.2 common commands that are implemented in the HP 3588A. An important property of all common commands is that you can send them without regard to a program message's position in the command tree. For more information on the analyzer's command tree, see chapter 3.

*CAL? query

Calibrates the analyzer and returns the result.

Query Syntax: `*CAL?`

Example Statements:
```
Output 719;"*CAL?"
Output 719;"*cal?"
```

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The analyzer performs a full calibration when you send this query. If the calibration completes without error, the analyzer returns 0. If the calibration fails, the analyzer returns 1.

This query is the same as the CAL:ALL? query.

*CLS command

Clears the Status Byte by emptying the error queue and clearing all event registers.

Command Syntax: `*CLS`

```
Example Statements: Output 719;"*Cls"
                    Output 719;"*CLS"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command clears the Status Byte register. It does so by emptying the error queue and clearing (setting to 0) all bits in the event registers of the following register sets:

- User Event.
- Device State.
- Questionable Power.
- Questionable Frequency.
- Limit Fail.
- Questionable Data.
- Event Status.
- Standard Operation.

In addition, *CLS cancels any preceding *OPC command or query. This ensures that bit 0 of the Standard Event register will not be set to 1 and that no response will be placed in the analyzer's output queue when pending overlapped commands are completed.

*CLS does not change the current state of enable registers or transition filters.

Note To guarantee that the Status Byte's Message Available and Master Summary Status bits will be cleared, send *CLS immediately following a Program Message Terminator.

See chapter 5 for more information on the Status Byte register.

*ESE

command/query

Sets bits in the Standard Event enable register.

Command Syntax: `*ESE {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:255
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"*ese 1"
                    Output 719;"*Ese 64"
```

Query Syntax: `*ESE?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: yes
Pass control required: no
```

Description:

This command allows you to set bits in the Standard Event enable register. Assign a decimal weight to each bit you want set (to 1) according to the following formula:

$$2^{(bit\_number)}$$

with acceptable values for bit_number being 0 through 7. Then add the weights and send the sum with this command.

When an enable register bit is set to 1, the corresponding bit of the Standard Event event register is enabled. All enabled bits are logically ORed to create the Standard Event summary, which reports to bit 5 of the Status Byte. Bit 5 is only set to 1 if both of the following are true:

- One or more bits in the Standard Event event register are set to 1.
- At least one set bit is enabled by a corresponding bit in the Standard Event enable register.

The setting last specified with *ESE is saved in nonvolatile memory. It can be recalled at power-up, depending on the setting of the Power-on Status Clear flag (set with *PSC). When the flag is 0 at power-up, all bits in the Standard Event enable register are set according to the saved *ESE value. When the flag is 1 at power-up, all bits in the Standard Event enable register are initialized to 0.

The query returns the current state of the Standard Event enable register. The state is returned as a sum of the decimal weights of all set bits.

For more information on the Standard Event register set, see chapter 5.

*ESR? query

Reads and clears the Standard Event event register.

Query Syntax: `*ESR?`

```
Example Statements: Output 719;"*ESR?"
                    Output 719;"*esr?"
```

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
             limits:  0:255
```

Attribute Summary:

```
Preset state: not defined
Overlapped: no
Pass control required: no
```

Description:

This query returns the current state of the Standard Event event register. The state is returned as a sum of the decimal weights of all set bits. The decimal weight for each bit is assigned according to the following formula:

$$2^{(bit\_number)}$$

with acceptable values for bit_number being 0 through 7.

The register is cleared after being read by this query.

A bit in this register is set to 1 when the condition it monitors becomes true. A set bit remains set, regardless of further changes in the condition it monitors, until one of the following occurs:

- You read the register with this query.
- You clear all event registers with the *CLS command.

For more information on the Standard Event register set, see chapter 5.

*IDN? query

Returns a string that uniquely identifies the analyzer.

Query Syntax: `*IDN?`

```
Example Statements: Output 719;"*Idn?"
                    Output 719;"*IDN?"
```

Return Format: `"<company>,<model>,<serial_num>,0"`

```
  <company> ::= HEWLETT-PACKARD
    <model> ::= 3588A
<serial_num> ::= 10 ASCII characters
```

Attribute Summary:

```
Preset state: instrument-dependent
Overlapped: no
Pass control required: no
```

Description:

The response to this query uniquely identifies your analyzer.

*OPC — command/query

Tells you when all pending overlapped commands have been completed.

Command Syntax: `*OPC`

Example Statements:
```
Output 719;"*opc"
Output 719;"*Opc?"
```

Query Syntax: `*OPC?`

Return Format: +1

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

Some commands are processed sequentially by the analyzer. A sequential command holds off the processing of subsequent commands until it has been completely processed. However, most commands do not hold off the processing of subsequent commands; they are referred to as overlapped commands.

The analyzer uses the No Pending Operation (NPO) flag to keep track of overlapped commands that are still pending (that is, not completed). The NPO flag is reset to 0 when an overlapped command is pending. It is set to 1 when no overlapped commands are pending. You cannot read the NPO flag directly, but you can use *OPC and *OPC? to tell when the flag is set to 1.

If you use *OPC, bit 0 of the Event Status event register is set to 1 when the NPO flag is set to 1. This allows the analyzer to generate a service request when all pending overlapped commands are completed (assuming you have enabled bit 0 of the Event Status register and bit 5 of the Status Byte register).

If you use *OPC?, 1 is placed in the output queue when the NPO flag is set to 1. This allows you to effectively pause the controller until all pending overlapped commands are completed. It must wait until the response is placed in the queue before it can continue.

Note

The *CLS and *RST commands cancel any preceding *OPC command or query. Pending overlapped commands are still completed, but you can no longer determine when. Two HP-IB bus management commands — Device Clear (DCL) and Selected Device Clear (SDC) — also cancel any preceding *OPC command or query.

*PCB command

Sets the pass-control-back address.

Command Syntax: `*PCB <number_1>[,<number_2>]`

```
<number_1> ::= MAX|MIN|an integer (NRf data)
               limits:  0:30
<number_2> ::= MAX|MIN|an integer (NRf data)
               limits:  0:30
```

```
Example Statements: Output 719;"*PCB 21"
                    Output 719;"*pcb 17"
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

Use this command to specify the address of your controller before you pass control of the HP-IB to the analyzer. When the analyzer completes the operation that required it to have control of the bus, it automatically passes control back to the controller at the specified address.

The optional second number is only used for controllers that support extended addressing. It is interpreted as the secondary address of the controller.

The address last specified with this command is saved in nonvolatile memory, so it is unaffected when you turn the analyzer off and on. It is also unaffected by the *RST command.

*PSC

command/query

Sets the state of the Power-on Status Clear flag.

Command Syntax: `*PSC {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  -32767:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"*Psc 0"
                    Output 719;"*PSC 1"
```

Query Syntax: `*PSC?`

Return Format: `+{0|1}`

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

This command lets you specify whether or not the Service Request enable register and the Event Status enable register should be cleared (all bits reset to 0) at power-up.

The settings of the Service Request enable register and the Event Status enable register are saved in nonvolatile memory when you turn the analyzer off. These settings can be recalled when you turn the analyzer on, but only if the Power-on Status Clear (PSC) flag is reset to 0. When the PSC flag is set to 1, the two enable registers are cleared at power-up. Use *PSC to specify the state of the PSC flag.

The number last specified with *PSC is saved in nonvolatile memory, so it is unaffected when you turn the analyzer off and back on. It is also unaffected by the *RST command.

If you want the analyzer to generate a service request at power-up, bit 7 of the Event Status event register and bit 5 of the Status Byte register must be enabled. This is only possible if the PSC flag is reset to 0.

The query returns the current state of the PSC flag.

*RST command

Executes a device reset.

Command Syntax: `*RST`

```
Example Statements: Output 719;"*rst"
                    Output 719;"*Rst"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command returns the analyzer to its preset state. In addition, it cancels any pending *OPC command or query.

Note

The preset state of each parameter is listed under the Attribute Summary of the associated command.

The following are not affected by this command:

- The state of the Power-on Status Clear flag.
- The state of all enable and transition registers.
- The HP-IB input and output queues.
- The time and date (SYST:TIME and SYST:DATE).
- The HP-IB address settings (SYST:COMM:GPIB:ADDR, PLOT:ADDR, and PRIN:ADDR).
- The HP-IB controller capability setting.
- The default disk selection (MMEM:MSI).
- Contents of limit, data, function, and constant registers.
- Contents of the RAM disks.
- Calibration constants.

*SRE

command/query

Sets bits in the Service Request enable register.

Command Syntax: `*SRE {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:255

 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"*SRE 12"
                    Output 719;"*sre 4"
```

Query Syntax: `*SRE?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

This command allows you to set bits in the Service Request enable register. Assign a decimal weight to each bit you want set (to 1) according to the following formula:

$$2^{(bit\_number)}$$

with acceptable values for bit_number being 0 through 7. Then add the weights and send the sum with this command.

Note

The analyzer ignores the setting you specify for bit 6 of the Service Request enable register. This is because the corresponding bit of the Status Byte register is always enabled.

The analyzer requests service from the active controller when one of the following occurs:

- A bit in the Status Byte register changes from 0 to 1 while the corresponding bit of the Service Request enable register is set to 1.

- A bit in the Service Request enable register changes from 0 to 1 while the corresponding bit of the Status Byte register is set to 1.

The setting last specified with *SRE is saved in nonvolatile memory. It can be recalled at power-up, depending on the setting of the Power-on Status Clear flag (set with *PSC). When the flag is 0 at power-up, all bits in the Service Request enable register are set according to the saved *SRE value. When the flag is 1 at power-up, all bits in the Service Request enable register are initialized to 0.

The query returns the current state of the Service Request enable register. The state is returned as a sum of the decimal weights of all set bits.

*STB? query

Reads the status byte register.

Query Syntax: `*STB?`

```
Example Statements: Output 719;"*Stb?"
                    Output 719;"*STB?"
```

Return Format: <number>

```
<number> ::= an integer (NR1 data)
             limits:  0:255
```

Attribute Summary:

```
Preset state: not defined
Overlapped: no
Pass control required: no
```

Description:

This query returns the current state of the Status Byte register. The state is returned as a sum of the decimal weights of all set bits. The decimal weight for each bit is assigned according to the following formula:

$$2^{(\text{bit\_number})}$$

with acceptable values for bit_number being 0 through 7.

The register is not cleared by this query. To clear the Status Byte register, you must send the *CLS command.

Bits in the Status Byte register are defined as follows:

- Bit 0 summarizes all enabled bits of the User Status event register.
- Bit 1 is reserved.
- Bit 2 summarizes all enabled bits of the Device State event register.
- Bit 3 summarizes all enabled bits of the Questionable Data event register.
- Bit 4 is the Message Available (MAV) bit. It is set whenever there is something in the analyzer's output queue.
- Bit 5 summarizes all enabled bits of the Event Status event register.
- Bit 6, when read with this query (*STB?), acts as the Master Summary Status (MSS) bit. It summarizes all enabled bits of the Status Byte register. (Bit 6 acts as the Request Service (RQS) bit when it is read by a serial poll.)
- Bit 7 summarizes all enabled bits of the Standard Operation event register.

For more information on the Status Byte register, see chapter 5.

*TRG

command

Triggers the analyzer if TRIG:SOUR is BUS.

Command Syntax: `*TRG`

```
Example Statements: Output 719;"*trg"
                    Output 719;"*Trg"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command triggers the analyzer if the following two conditions are met:

- The HP-IB is designated as the trigger source. (Send the TRIG:SOUR BUS command.)
- The analyzer is waiting to trigger. (Bit 5 of the Standard Operation condition register must be set.)

The *TRG command has the same effect as TRIG:IMM. It also has the same effect as the HP-IB bus management command Group Execute Trigger (GET) with the following exception: *TRG is sent to the input queue and processed in the order received, but GET is processed immediately, even if the input queue contains other commands.

*TST? query

Tests the analyzer hardware and returns the result.

Query Syntax: `*TST?`

Example Statements:
```
Output 719;"*TST?"
Output 719;"*tst?"
```

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The analyzer's self-test performs a full calibration and then compares the resulting trace to specified limits. If the trace is within the limits, the analyzer returns 0. If the trace exceeds the limits, the analyzer returns 1.

*WAI — command

Holds off processing of subsequent commands until all preceding commands have been processed.

Command Syntax: `*WAI`

```
Example Statements: Output 719;"*WAI"
                    Output 719;"*wai"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

Some commands are processed sequentially by the analyzer. A sequential command holds off the processing of any subsequent commands until it has been completely processed. However, most commands do not hold off the processing of subsequent commands; they are referred to as overlapped commands. *WAI ensures that overlapped commands will be completely processed before subsequent commands (those sent after *WAI) are processed.

9

ABORt Subsystem

This subsystem contains a single command—ABORt—which is used to immediately stop the current measurement.

ABORt

command

Immediately stops the measurement in progress and starts a new one.

Command Syntax: `ABORt`

Example Statements:
```
Output 719;"Abort"
Output 719;"ABOR"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

Although this command aborts any measurement in progress, it does not set INIT:CONT to OFF (as defined in the TMSL standard). The HP 3588A does not support the OFF state for INIT:CONT. As a result, ABOR has the same effect as SENS:REST—it aborts the current measurement *and* starts a new one.

Both ABOR;INIT:IMM and SENS:REST serve a special synchronizing function. When you send either of these program messages to restart a measurement, the analyzer's No Pending Operation (NPO) flag is not set to 1 until the measurement is complete. The two commands that test the state of this flag—*WAI and *OPC—allow you to hold off subsequent actions until the measurement is complete. See "Synchronization" in chapter 2 for more information on the NPO flag.

Note

When video averaging is enabled (AVER:TYPE RMS and AVER:STAT ON), the NPO flag is not set to 1 until *n* measurements have been combined into one trace. You specify the value of *n* with the AVER:COUN command.

10

ARM Subsystem

The ARM subsystem contains two commands that control the analyzer's trigger arming functions. One command selects the type of arming (automatic or manual). The other command arms the trigger (when manual arming is selected). See the TRIGger subsystem for commands that control other trigger functions.

ARM[:IMMediate]

command

Arms the trigger if ARM:SOUR is MAN.

Command Syntax: `ARM[:IMMediate]`

```
Example Statements: Output 719;"arm"
                    Output 719;"Arm:Imm"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

Two conditions must be met before this command can arm the trigger:

- Manual arming must be selected (ARM:SOUR is MAN).
- Bit 1 (Settling) *or* bit 6 (Waiting for ARM) of the Standard Operation condition register must be set to 1.

ARM:IMM is ignored at all other times.

ARM:SOURce

command/query

Specifies whether arming is automatic or manual.

Command Syntax: `ARM:SOURce {IMMediate|MANual}`

Example Statements:
```
Output 719;"ARM:SOURCE IMMEDIATE"
Output 719;"arm:sour man"
```

Query Syntax: `ARM:SOURce?`

Return Format: `IMM|MAN`

Attribute Summary:
```
Preset state: IMM
Overlapped: yes
Pass control required: no
```

Description:

Send IMM to select automatic arming. Send MAN to select manual arming.

When automatic arming is selected, the analyzer waits for the hardware to settle, and then waits for a trigger signal (specified by TRIG:SOUR) before starting a measurement. When manual arming is selected, the analyzer waits for the hardware to settle, waits for ARM:IMM, and then waits for a trigger signal before starting a measurement.

Note

If you send ARM:SOUR MAN and TRIG:SOUR IMM, then you can use ARM:IMM to trigger a single sweep.

11

AVERage Subsystem

The AVERage subsystem contains commands that define how the results of several measurements will be combined in one trace.

AVERage:COUNt

command/query

Specifies a count and a weighting factor for averaged measurement data.

Command Syntax: `AVERage:COUNt {<number>|<step>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  1:1024
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Aver:Coun 10"
                    Output 719;"AVERAGE:COUNT 5"
```

Query Syntax: `AVERage:COUNt?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: +10
Overlapped: yes
Pass control required: no
```

Description:

The value in AVER:COUN is only used when the following conditions are met:

- AVER:STAT is ON (1).
- AVER:TYPE is VID (RMS).

In its role as a counter, AVER:COUN determines how many measurement results will be combined in one trace before the No Pending Operation (NPO) flag is set to 1 for SENS:REST. This lets you use *OPC to determine when the specified number of measurement results have been combined. (See “Synchronization” in chapter 2 for more information on the NPO flag.)

In its role as a weighting factor, AVER:COUN (*N*) determines how the results of the current measurement (*new* data) will be combined with the averaged trace (*old* data). Data is combined, point by point, according to the following formula:

$$\left(\frac{1}{N} \times \text{new}\right) + \left(\frac{N-1}{N} \times \text{old}\right)$$

Note

Averaging does not stop after the count is reached.

AVERage[:STATe] — command/query

Turns the selected averaging function (AVER:TYPE) on and off.

Command Syntax: `AVERage[:STATe] {OFF|0|ON|1}`

Example Statements:
```
Output 719;"average:state off"
Output 719;"Aver 1"
```

Query Syntax: `AVERage[:STATe]?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +0
Overlapped: yes
Pass control required: no
```

Description:

When you select OFF, each trace represents the results of a single measurement. When you select ON, each trace represents the combined results of several measurements, and the averaging function specified in AVER:TYPE determines how results are combined.

When averaging is on and AVER:TYPE is RMS, SENS:REST does not set the No Pending Operation (NPO) flag to 1 until the specified number (AVER:COUN) of measurement results have been combined. When averaging is on and AVER:TYPE is MAX (or when averaging is off), SENS:REST sets the NPO flag to 1 each time a measurement is completed. See "Synchronization" in chapter 2 for more information on the NPO flag.

Note

Trigger conditions must be met for each measurement—even when averaging is turned on.

AVERage:TCONtrol

command/query

Specifies the analyzer's behavior after the count (AVER:COUN) is reached.

Command Syntax: `AVERage:TCONtrol {EXPonential}`

Example Statements:
```
Output 719;"AVER:TCON EXP"
Output 719;"average:tcontrol exponential"
```

Query Syntax: `AVERage:TCONtrol?`

Return Format: `EXP`

Attribute Summary:
```
Preset state: EXP
Overlapped: yes
Pass control required: no
```

Description:

EXP is the only valid option for the HP 3588A. It indicates that averaging doesn't stop when the count (AVER:COUN) is reached. This command is only included for TMSL compatibility.

AVERage:TYPE

command/query

Selects a method for combining the results of several measurements.

Command Syntax: `AVERage:TYPE {RMS|MAX|VIDeo|PEAK}`

Example Statements:

```
Output 719;"Average:Type Max"
Output 719;"AVER:TYPE VID"
```

Query Syntax: `AVERage:TYPE?`

Return Format: `RMS|MAX`

Attribute Summary:

```
Preset state: RMS
Overlapped: yes
Pass control required: no
```

Description:

RMS and VID both select the analyzer's video averaging function. MAX and PEAK both select the analyzer's peak hold function. Use the TMSL-supported RMS and MAX parameters if you plan to use your program with other TMSL instruments.

To enable the selected averaging function, you must set AVER:STAT to ON.

12

CALCulate Subsystem

The CALCulate subsystem contains commands that control the processing of measurement data. The commands let you select a coordinate system for display of the measurement data, define trace math functions and constants, and dump coordinate transformed data to your controller. The following block diagram shows you how measurement data is processed:

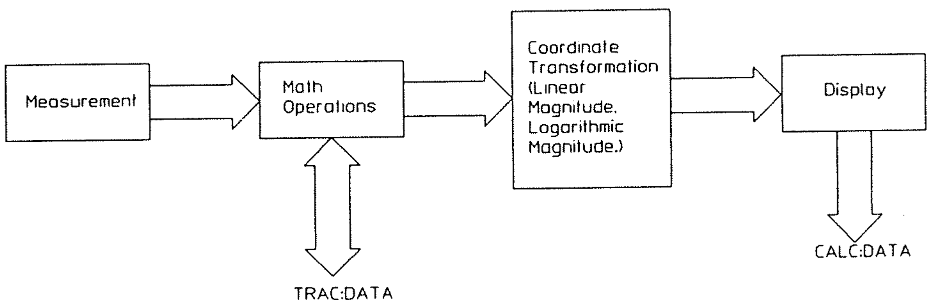


Figure 12-1. Flow of Measurement Data

After measurement data is collected, any specified math operations are performed. Data is then transformed into the specified coordinate system and sent to the display. TRAC:DATA gives you access to the raw measurement data after math operations have been performed. CALC:DATA gives you access to the display data—after the coordinate transformation.

Note

You can take measurement data out of the analyzer with either TRAC:DATA or CALC:DATA, but you can only put it back into the analyzer with TRAC:DATA.

The CALCulate mnemonic contains an optional trace specifier: [1|2]. To direct a command to trace A, omit the specifier or use 1. To direct a command to trace B, use 2. Commands that are not trace-specific—like CALC:MATH:EXPR—ignore the specifier.

CALCulate[1|2]:DATA? **query**

Returns trace data that is transformed to the current coordinate system (CALC:FORM).

Query Syntax: `CALCulate[1|2]:DATA?`

```
Example Statements: Output 719;"calc2:data?"
                    Output 719;"Calculatel:Data?"
```

Return Format: `<block>`

When data is ASCII-encoded, (FORM ASC) <block> takes the following form:

```
<block> ::= <point>{,<point>}...
<point> ::= y-axis value for 1 of the 401 points that make up
            a trace
            limits:  -9.9E37:9.9E37
```

When data is binary-encoded, (FORM REAL) <block> takes the following form:

```
        <block> ::= #<byte><length_bytes>{<point>}...
         <byte> ::= one ASCII-encoded byte specifying the number of length
                    bytes to follow
<length_bytes> ::= ASCII-encoded bytes specifying the number of data
                    bytes to follow
        <point> ::= y-axis value for 1 of the 401 points that make up
                    a trace
                    limits:  -9.9E37:9.9E37
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns a block of coordinate-transformed trace data. The block is returned as a series of 401 amplitude values. The unit for these values is the same as the reference level unit (returned with DISP:Y:SCAL:MAX? UNIT).

You cannot return trace data to the display with CALC:DATA, because it has no command form. Use TRAC:DATA if you want to read and write trace data. (See the introduction to this chapter for more information.)

CALCulate[1|2]:FORMat **command/query**

Selects a coordinate system for displaying and dumping trace data.

Command Syntax: `CALCulate[1|2]:FORMat {MLINear|MLOGarithmic|NONE}`

Example Statements:

```
Output 719;"CALCULATE2:FORMAT MLOGARITHMIC"
Output 719;"calc1:form mlin"
```

Query Syntax: `CALCulate[1|2]:FORMat?`

Return Format: `MLIN|MLOG`

Attribute Summary:

```
Preset state: MLOG (both traces)
Overlapped: yes
Pass control required: no
```

Description:

Send MLIN to select a coordinate system that displays linear magnitude amplitude data versus frequency (or time). Send MLOG to select a coordinate system that displays logarithmic magnitude amplitude data versus frequency (or time).

If you query the analyzer with CALC:DATA?, the values returned are already transformed to the coordinate system you specify with CALC:FORM.

CALCulate[1|2]:MATH:CONStant — command/query

Loads a number into one of the constant registers.

Command Syntax: `CALCulate[1|2]:MATH:CONStant <const>,{<number>|<bound>}`

```
<const> ::= {K1|K2|K3|K4|K5}
<number> ::= a real number (NRf data)
             limits:   -9.9E37:9.9E37
<bound> ::= MAX|MIN
```

Example Statements:

```
Output 719;"Calc:Math:Cons K1,.707"
Output 719;"CALCULATE1:MATH:CONSTANT K5,1.414"
```

Query Syntax: `CALCulate[1|2]:MATH:CONStant? <const>`

Return Format: `<number>`

```
<number> ::= a real number (NR2 or NR3 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: yes
Pass control required: no
```

Description:

To use a constant in a trace math function, you must first load it into one of the analyzer's five constant registers: K1 through K5. You can then include the constant register's name at the appropriate place in your function. (Functions are defined with the CALC:MATH:EXPR command.)

CALCulate[1|2]:MATH:DATA **command/query**

Loads a complete set of math definitions.

Command Syntax: `CALCulate[1|2]:MATH:DATA <block>`

```
          <block> ::= #<byte>[<length_bytes>]<data_bytes>
           <byte> ::= one ASCII-encoded byte specifying the number of length
                      bytes to follow
   <length_bytes> ::= ASCII-encoded bytes specifying the number of data
                      bytes to follow
     <data_bytes> ::= the bytes that make up a complete set of math
                      definitions

Example Statements: Output 719;"calculate2:math:data?"
                    Output 719;"Calc:Math:Data?"
```

Query Syntax: `CALCulate[1|2]:MATH:DATA?`

Return Format: `<block>`

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

This command allows you to transfer a complete set of math definitions—the same information contained in a math file—between the analyzer and your controller. This allows you to store a set of math definitions on your controller's file system. The definitions cannot be altered.

When you transfer a set of math definitions to the analyzer, you can use either the definite or the indefinite length block syntax. When the analyzer returns the set to a controller, it always uses the definite length block syntax. See "Block Data" in chapter 4 for more information.

CALCulate[1|2]:MATH[:EXPRession] command/query

Loads an expression into one of the constant registers.

Command Syntax:

```
CALCulate[1|2]:MATH[:EXPRession] <func>,<expr>

          <func> ::= {F1|F2|F3|F4|F5}
          <expr> ::= ([<expr_element>]...)
  <expr_element> ::= SPEC, D1:D8, K1:K5, F1:F5, SQRT(NBW), (, ), +, -, *, /
```

Example Statements:

```
Output 719;"CALC:MATH F3,(SPEC/D1)"
Output 719;"calculate1:math:expression f2,(spec*k2)"
```

Query Syntax: `CALCulate[1|2]:MATH[:EXPRession]? <func>`

Return Format: `"<expr>"`

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: yes
Pass control required: no
```

Description:

Before you can display the results of a trace math function, you must load the function into one of the analyzer's five function registers: F1 through F5. Once you have loaded the function register with CALC:MATH:EXPR, you can display the results with DISP:RES.

You define trace math functions using operands (SPEC, D1:D8, K1:K5, F1:F5, SQRT(NBW)) and operators (+, −, *, /). You combine these elements according to the rules of standard algebraic notation. Use parentheses to control the order of operations.

Refer to the help text or the *HP 3588A Operating Manual* for more information on trace math functions.

13

CALibration Subsystem

The CALibration subsystem contains the commands that control the analyzer's self-calibration functions. You can initiate a single calibration or enable the autocalibration function. After calibration, you can display the calibration data for the active trace.

This subsystem also contains a command that allows you to enable and disable the analyzer's oversweeping function.

CALibration[:ALL]? query

Calibrates the analyzer and returns the result.

Query Syntax: `CALibration[:ALL]?`

Example Statements:
```
Output 719;"Calibration:All?"
Output 719;"CAL?"
```

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The analyzer performs a full calibration when you send this query. If the calibration completes without error, the analyzer returns 0. If the calibration fails, the analyzer returns 1.

This query is the same as the *CAL? query.

CALibration:AUTO

command/query

Calibrates the analyzer or sets the state of the autocalibration function.

Command Syntax: `CALibration:AUTO {OFF|0|ON|1|ONCE}`

```
Example Statements: Output 719;"cal:auto 0"
                    Output 719;"Calibration:Auto Off"
```

Query Syntax: `CALibration:AUTO?`

Return Format: `+{0|1}`

Attribute Summary:

```
Preset state: +1
Overlapped: yes
Pass control required: no
```

Description:

Send ON to enable the analyzer's autocalibration function, OFF to disable it. This function calibrates the analyzer several times during the first hour of operation and once per hour thereafter.

Send ONCE to initiate a single calibration.

CALibration:CORRection:SRATe — command/query

Turns oversweeping on and off.

Command Syntax: `CALibration:CORRection:SRATe {OFF|0|ON|1}`

Example Statements:
```
Output 719;"CALIBRATION:CORRECTION:SRATE ON"
Output 719;"cal:corr:srat 0"
```

Query Syntax: `CALibration:CORRection:SRATe?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: yes
Pass control required: no
```

Description:

The analyzer provides calibrated measurement results for sweep rates at or below the following limits (given in Hz/s):

- Oversweep on: $RBW^2 \times 2$
- Oversweep off: $RBW^2 \div 2$

where RBW is the setting of SENS:BAND:RES (in Hz)

14

DIAGnostics Subsystem

Commands in the DIAGnostics subsystem access functions that should only be used as described in the *HP 3588A Performance Test Guide*. Refer to that manual for more information on these functions.

DIAGnostics:SOURce:PAD:TEN

command/query

Switches the source's 10 dB attenuator in and out.

Command Syntax: `DIAGnostics:SOURce:PAD:TEN {IN|OUT}`

Example Statements:
```
Output 719;"diagnostics:source:pad:ten out"
Output 719;"Diag:Sour:Pad:Ten Out"
```

Query Syntax: `DIAGnostics:SOURce:PAD:TEN?`

Return Format: `IN|OUT`

Attribute Summary:
```
Preset state: OUT
Overlapped: yes
Pass control required: no
```

Description:

Use this command only as directed in the *HP 3588A Performance Test Guide*.

DIAGnostics:SOURce:PADA:TWENty **command/query**

Switches the source's first 20 dB attenuator in and out.

Command Syntax: `DIAGnostics:SOURce:PADA:TWENty {IN|OUT}`

Example Statements:
```
Output 719;"DIAG:SOUR:PADA:TWEN OUT"
Output 719;"diagnostics:source:pada:twenty in"
```

Query Syntax: `DIAGnostics:SOURce:PADA:TWENty?`

Return Format: `IN|OUT`

Attribute Summary:
```
Preset state: OUT
Overlapped: yes
Pass control required: no
```

Description:

Use this command only as directed in the *HP 3588A Performance Test Guide.*

DIAGnostics:SOURce:PADB:TWENty command/query

Switches the source's second 20 dB attenuator in and out.

Command Syntax: `DIAGnostics:SOURce:PADB:TWENty {IN|OUT}`

Example Statements:
```
Output 719;"Diagnostics:Source:Padb:Twenty In"
Output 719;"DIAG:SOUR:PADB:TWEN OUT"
```

Query Syntax: `DIAGnostics:SOURce:PADB:TWENty?`

Return Format: `IN|OUT`

Attribute Summary:
```
Preset state: OUT
Overlapped: yes
Pass control required: no
```

Description:

Use this command only as directed in the *HP 3588A Performance Test Guide*.

15

DISPlay Subsystem

DISPlay is one of two subsystems that control the analyzer's presentation of data on its front-panel display—the other is SCReen. DISPlay also allows you to define trace limits and control limit testing.

The DISPlay mnemonic contains an optional trace specifier: [1|2]. To direct a command to trace A, omit the specifier or use 1. To direct a command to trace B, use 2. Commands that are not trace-specific—like DISP:PART:CLE—ignore the specifier.

DISPlay[1|2]:LIMit:BEEP

command/query

Turns the limit-fail beeper on and off.

Command Syntax: `DISPlay[1|2]:LIMit:BEEP {OFF|0|ON|1}`

Example Statements:
```
Output 719;"disp:lim:beep 1"
Output 719;"Display2:Limit:Beep Off"
```

Query Syntax: `DISPlay[1|2]:LIMit:BEEP?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +0 (both displays)
Overlapped: yes
Pass control required: no
```

Description:

The limit-fail beeper emits an audible tone when all of the following conditions are met:

- DISP:LIM:BEEP is ON.
- SYST:BEEP:STAT is ON.
- DISP:LIM:STAT is ON.
- The trace falls outside its current limits.

You can use DISP:LIM:LOW:SEGM and DISP:LIM:UPP:SEGM to define a trace's current limits via the HP-IB.

DISPlay[1|2]:LIMit:LINE

command/query

Turns limit lines on and off in the specified display.

Command Syntax: `DISPlay[1|2]:LIMit:LINE {OFF|0|ON|1}`

Example Statements:

```
Output 719;"Displ:Lim:Line Off"
Output 719;"DISPLAY:LIMIT:LINE 1"
```

Query Syntax: `DISPlay[1|2]:LIMit:LINE?`

Return Format: `+{0|1}`

Attribute Summary:

```
Preset state: +0 (both displays)
Overlapped: yes
Pass control required: no
```

Description:

Sending DISP:LIM:LINE ON only enables the display of limit lines in the specified trace. To test the trace against those limits, you must send DISP:LIM:STAT ON.

Note A trace can be evaluated against limits even when limit lines are not displayed.

You can use DISP:LIM:LOW:SEGM and DISP:LIM:UPP:SEGM to define limit lines via the HP-IB.

DISPlay[1|2]:LIMit:LOWer:DATA command/query

Loads a complete lower limit line into the specified display.

Command Syntax: `DISPlay[1|2]:LIMit:LOWer:DATA <block>`

```
         <block> ::= #<byte>[<length_bytes>]<data_bytes>
          <byte> ::= one ASCII-encoded byte specifying the number of length
                     bytes to follow
  <length_bytes> ::= ASCII-encoded bytes specifying the number of data
                     bytes to follow
    <data_bytes> ::= the bytes that make up a complete lower limit line
```

```
Example Statements: Output 719;"display2:limit:lower:data?"
                    Output 719;"Disp:Lim:Low:Data?"
```

Query Syntax: `DISPlay[1|2]:LIMit:LOWer:DATA?`

Return Format: `<block>`

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

This command transfers a complete lower limit line—the same information contained in a lower limit file—between the analyzer and your controller. This allows you to store a complete lower limit on your controller's file system. The limit cannot be altered.

When you transfer a complete lower limit to the analyzer, you can use either the definite or the indefinite length block syntax. When the analyzer returns the limit to a controller, it always uses the definite length block syntax. See "Block Data" in chapter 4 for more information.

DISPlay[1|2]:LIMit:LOWer:DELete

command

Deletes the lower limit line from the specified display.

Command Syntax: `DISPlay[1|2]:LIMit:LOWer:DELete`

Example Statements:
```
Output 719;"DISPLAY1:LIMIT:LOWER:DELETE"
Output 719;"disp2:lim:low:del"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

To delete a lower limit, send DISP:LIM:LOW:DEL. To delete an upper limit, send DISP:LIM:UPP:DEL.

DISPlay[1|2]:LIMit:LOWer:MOVE command/query

Moves the lower limit line up or down in the specified trace.

Command Syntax: `DISPlay[1|2]:LIMit:LOWer:MOVE {<number>|<step>|<bound>}`

```
<number> ::= a real number (NRf data)
             limits:   -9.9E37:9.9E37
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"DISPLAY1:LIMIT:LOWER:MOVE 3"
                    Output 719;"disp:lim:low:move -12"
```

Query Syntax: `DISPlay[1|2]:LIMit:LOWer:MOVE?`

Return Format: `<number>`

```
<number> ::= a real number (NR2 or NR3 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: yes
Pass control required: no
```

Description:

DISP:LIM:LOW:MOVE specifies a vertical offset for every segment in a lower limit. The offset is referenced to the limit's original y-axis position. The offset is unitless, so it assumes the current vertical/division unit (returned with DISP:Y:SCAL:PDIV? UNIT).

DISPlay[1|2]:LIMit:LOWer:REPort? query

Returns the frequency, amplitude, and failure value for all points failing the lower limit test.

Query Syntax: `DISPlay[1|2]:LIMit:LOWer:REPort?`

Example Statements:
```
Output 719;"DISP2:LIM:LOW:REP?"
Output 719;"display:limit:lower:report?"
```

Return Format: <block>

When data is ASCII-encoded, (FORM ASC) <block> takes the following form:

```
<block> ::= [<point>[,<point>]...]
<point> ::= <frequency>,<amplitude>,<failed_by>
```

When data is binary-encoded, (FORM REAL) <block> takes the following form:

```
       <block> ::= #<byte><length_bytes>[<point>]...
        <byte> ::= one ASCII-encoded byte specifying the number of length
                   bytes to follow
<length_bytes> ::= ASCII-encoded bytes specifying the number of data
                   bytes to follow
       <point> ::= <frequency><amplitude><failed_by>
```

The following definitions apply to both ASCII- and binary-encoded data.

```
<frequency> ::= a real number (frequency of the failed point)
                limits:  0.0:150.0E6 (Hz)
<amplitude> ::= a real number (amplitude of the failed point)
                limits:  -9.9E37:9.9E37
<failed_by> ::= a real number (amplitude offset from limit)
                limits:  -9.9E37:9.9E37
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The unit for returned amplitude values is the same as the current reference level unit (returned with DISP:Y:SCAL:MAX? UNIT). The unit for returned failure values is the same as the current vertical/division unit (returned with DISP:Y:SCAL:PDIV? UNIT).

No data is returned if limit testing is disabled (DISP:LIM:STAT OFF) or if all trace points are above the specified lower limit.

DISPlay[1|2]:LIMit:LOWer:SEGMent — command/query

Loads one or more segments of a lower limit line into the specified display.

Command Syntax: `DISPlay[1|2]:LIMit:LOWer:SEGMent <block>`

When data is ASCII-encoded, (FORM ASC) <block> takes the following form:

<block> ::= <segment>[,<segment>]...
<segment> ::= <start_freq>,<start_ampl>,<stop_freq>,<stop_ampl>

When data is binary-encoded, (FORM REAL) <block> takes the following form:

<block> ::= #<byte>[<length_bytes>]<segment>[<segment>]...
<byte> ::= one ASCII-encoded byte specifying the number of length bytes to follow
<length_bytes> ::= ASCII-encoded bytes specifying the number of data bytes to follow
<segment> ::= <start_freq><start_ampl><stop_freq><stop_ampl>

The following definitions apply to both ASCII- and binary-encoded data.

```
  <start_freq> ::= a real number
                   limits:  0.0:150.0E6 (Hz)
  <start_ampl> ::= a real number
                   limits:  -9.9E37:9.9E37
   <stop_freq> ::= a real number
                   limits:  0.0:150.0E6 (Hz)
   <stop_ampl> ::= a real number
                   limits:  -9.9E37:9.9E37
```

```
Example Statements: Output 719;"Display1:Limit:Lower:Segment 7e6,0,10e6,5"
                    Output 719;"DISP:LIM:LOW:SEGM 10E6,5,13E6,0,13E6,0,16e6,-5"
```

Query Syntax: `DISPlay[1|2]:LIMit:LOWer:SEGMent?`

Return Format: `<block>`

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: yes
Pass control required: no
```

Description:

The amplitude values you send with each segment are unitless, so they assume the current reference level unit (returned with DISP:Y:SCAL:MAX? UNIT).

The analyzer doesn't clear the previous lower limit definition when you send new segments—it only overwrites those portions of the limit redefined by the new segments. Send DISP:LIM:LOW:DEL if you want to clear the previous limit.

DISPlay[1|2]:LIMit:MODE command/query

Specifies how limit lines should respond to changes in the reference level setting.

Command Syntax: `DISPlay[1|2]:LIMit:MODE {ABSolute|RELative}`

Example Statements:
```
Output 719;"disp2:lim:mode rel"
Output 719;"Display:Limit:Mode Absolute"
```

Query Syntax: `DISPlay[1|2]:LIMit:MODE?`

Return Format: `ABS|REL`

Attribute Summary:
```
Preset state: ABS (both displays)
Overlapped: yes
Pass control required: no
```

Description:

When you select ABS, the absolute y-axis values of limit lines are held constant when the reference level changes. When you select REL, the y-axis offsets between limit lines and the reference level are held constant when the reference level changes. (You change the reference level with the DISP:Y:SCAL:MAX command.)

DISPlay[1|2]:LIMit:RESult? query

Returns the result of the last limit test (passed or failed).

Query Syntax: `DISPlay[1|2]:LIMit:RESult?`

Example Statements:

```
Output 719;"Disp2:Lim:Res?"
Output 719;"DISPLAY:LIMIT:RESULT?"
```

Return Format: `OFF|UND|PASS|FAIL`

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query only returns PASS or FAIL if the limit testing is turned on (DISP:LIM:STAT ON) and a limit is defined for the specified trace. If limit testing is turned off, this query returns OFF. If no limit is defined, this query returns UND (undefined).

You can use DISP:LIM:LOW:SEGM and DISP:LIM:UPP:SEGM to define limits via the HP-IB.

DISPlay[1|2]:LIMit:STATe

command/query

Turns limit testing on and off.

Command Syntax: `DISPlay[1|2]:LIMit:STATe {OFF|0|ON|1}`

Example Statements:
```
Output 719;"display2:limit:state on"
Output 719;"Disp:Lim:Stat 1"
```

Query Syntax: `DISPlay[1|2]:LIMit:STATe?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +0 (both displays)
Overlapped: yes
Pass control required: no
```

Description:

When limit testing is on, the current trace is evaluated against the limits defined in its upper and lower limit registers. You can load these registers via the HP-IB using the DISP:LIM:LOW:SEGM and DISP:LIM:UPP:SEGM commands.

To determine whether or not a trace is within the specified limits, you can send the DISP:LIM:RES query or monitor bits in the Limit Fail condition register. To return failed points, you can send the DISP:LIM:LOW:REP and DISP:LIM:UPP:REP queries.

Note

Limit lines are not automatically displayed when limit testing is enabled. To display limits, you must send DISP:LIM:LINE ON.

DISPlay[1|2]:LIMit:UPPer:DATA

command/query

Loads a complete upper limit line into the specified display.

Command Syntax: `DISPlay[1|2]:LIMit:UPPer:DATA <block>`

```
<block> ::= #<byte>[<length_bytes>]<data_bytes>
<byte> ::= one ASCII-encoded byte specifying the number of length
           bytes to follow
<length_bytes> ::= ASCII-encoded bytes specifying the number of data
                   bytes to follow
<data_bytes> ::= the bytes that make up a complete upper limit line
```

```
Example Statements: Output 719;"DISP1:LIM:UPP:DATA?"
                    Output 719;"display2:limit:upper:data?"
```

Query Syntax: `DISPlay[1|2]:LIMit:UPPer:DATA?`

Return Format: `<block>`

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

This command just transfers a complete upper limit line—the same information contained in a upper limit file—between the analyzer and your controller. This allows you to store a complete upper limit on your controller's file system. The limit cannot be altered.

When you transfer a complete upper limit to the analyzer, you can use either the definite or the indefinite length block syntax. When the analyzer returns the limit to a controller, it always uses the definite length block syntax. See "Block Data" in chapter 4 for more information.

DISPlay[1|2]:LIMit:UPPer:DELete

command

Deletes the upper limit line from the specified display.

Command Syntax: `DISPlay[1|2]:LIMit:UPPer:DELete`

Example Statements:

```
Output 719;"DISPLAY1:LIMIT:UPPER:DELETE"
Output 719;"disp2:lim:upp:del"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

To delete an upper limit, send DISP:LIM:UPP:DEL. To delete a lower limit, send DISP:LIM:LOW:DEL.

DISPlay[1|2]:LIMit:UPPer:MOVE

command/query

Moves the upper limit line up or down in the specified trace.

Command Syntax: `DISPlay[1|2]:LIMit:UPPer:MOVE {<number>|<step>|<bound>}`

```
<number> ::= a real number (NRf data)
             limits:  -9.9E37:9.9E37
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"DISPLAY1:LIMIT:UPPER:MOVE 3"
                    Output 719;"disp:lim:upp:move -12"
```

Query Syntax: `DISPlay[1|2]:LIMit:UPPer:MOVE?`

Return Format: `<number>`

```
<number> ::= a real number (NR2 or NR3 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: yes
Pass control required: no
```

Description:

DISP:LIM:UPP:MOVE specifies a vertical offset for every segment in an upper limit. The offset is referenced to the limit's original y-axis position. The offset is unitless, so it assumes the current vertical/division unit (returned with DISP:Y:SCAL:PDIV? UNIT).

DISPlay[1|2]:LIMit:UPPer:REPort? query

Returns the frequency, amplitude, and failure value for all points failing the upper limit test.

Query Syntax: `DISPlay[1|2]:LIMit:UPPer:REPort?`

```
Example Statements: Output 719;"Display2:Limit:Upper:Report?"
                    Output 719;"DISP:LIM:UPP:REP?"
```

Return Format: <block>

When data is ASCII-encoded, (FORM ASC) <block> takes the following form:

```
<block> ::= [<point>[,<point>]...]
<point> ::= <frequency>,<amplitude>,<failed_by>
```

When data is binary-encoded, (FORM REAL) <block> takes the following form:

```
        <block> ::= #<byte><length_bytes>[<point>]...
         <byte> ::= one ASCII-encoded byte specifying the number of length
                    bytes to follow
 <length_bytes> ::= ASCII-encoded bytes specifying the number of data
                    bytes to follow
        <point> ::= <frequency><amplitude><failed_by>
```

The following definitions apply to both ASCII- and binary-encoded data.

```
<frequency> ::= a real number (frequency of the failed point)
                limits:  0.0:150.0E6 (Hz)
<amplitude> ::= a real number (amplitude of the failed point)
                limits:  -9.9E37:9.9E37
<failed_by> ::= a real number (amplitude offset from limit)
                limits:  -9.9E37:9.9E37
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The unit for returned amplitude values is the same as the current reference level unit (returned with DISP:Y:SCAL:MAX? UNIT). The unit for returned failure values is the same as the current vertical/division unit (returned with DISP:Y:SCAL:PDIV? UNIT).

No data is returned if limit testing is disabled (DISP:LIM:STAT OFF) or if all trace points are below the specified upper limit.

DISPlay[1|2]:LIMit:UPPer:SEGMent — command/query

Loads one or more segments of an upper limit line into the specified display.

Command Syntax: `DISPlay[1|2]:LIMit:UPPer:SEGMent <block>`

When data is ASCII-encoded, (FORM ASC) <block> takes the following form:

<block> ::= <segment>[,<segment>]...
<segment> ::= <start_freq>,<start_ampl>,<stop_freq>,<stop_ampl>

When data is binary-encoded, (FORM REAL) <block> takes the following form:

<block> ::= #<byte><length_bytes><segment>[<segment>]...
<byte> ::= one ASCII-encoded byte specifying the number of length bytes to follow
<length_bytes> ::= ASCII-encoded bytes specifying the number of data bytes to follow
<segment> ::= <start_freq><start_ampl><stop_freq><stop_ampl>

The following definitions apply to both ASCII- and binary-encoded data.

```
 <start_freq> ::= a real number
                  limits:  0.0:150.0E6 (Hz)
 <start_ampl> ::= a real number
                  limits:  -9.9E37:9.9E37
  <stop_freq> ::= a real number
                  limits:  0.0:150.0E6 (Hz)
  <stop_ampl> ::= a real number
                  limits:  -9.9E37:9.9E37
```

```
Example Statements: Output 719;"DISP:LIM:UPP:SEGM 10E6,5,13E6,0,13E6,0,16E6,-5"
                    Output 719;"Display2:Limit:Upper:Segment 7e6,0,10e6,5"
```

Query Syntax: `DISPlay[1|2]:LIMit:UPPer:SEGMent?`

Return Format: `<block>`

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: yes
Pass control required: no
```

Description:

The amplitude values you send with each segment are unitless, so they assume the current reference level unit (returned with DISP:Y:SCAL:MAX? UNIT).

The analyzer doesn't clear the previous upper limit definition when you send new segments—it only overwrites those portions of the limit redefined by the new segments. Send DISP:LIM:UPP:DEL if you want to clear the previous limit.

DISPlay[1|2]:PARTition

command/query

Select the portion of the analyzer's screen to be used for HP Instrument BASIC program output.

Command Syntax: `DISPlay[1|2]:PARTition {OFF|0|FULL|UPPer|LOWer}`

Example Statements:
```
Output 719;"DISPLAY:PARTITION FULL"
Output 719;"disp:part 0"
```

Query Syntax: `DISPlay[1|2]:PARTition?`

Return Format: `OFF|FULL|UPP|LOW`

Attribute Summary:
```
Preset state: OFF
Overlapped: yes
Pass control required: no
```

Description:

OFF allocates no display area for program output. FULL allocates the entire trace display area. UPP allocates that portion of the trace display area normally used by the *upper* trace (when SCR:FORM is ULOW). LOW allocates that portion of the trace display area normally used by the *lower* trace.

This command is only valid if HP Instrument BASIC is installed.

DISPlay[1|2]:PARTition:CLEar

command

Clear the portion of the analyzer's screen allocated to HP Instrument BASIC program output.

Command Syntax: `DISPlay[1|2]:PARTition:CLEar`

Example Statements:

```
Output 719;"Disp:Part:Cle"
Output 719;"DISPLAY2:PARTITION:CLEAR"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

You allocate the portion of the screen cleared by this command with the DISP:PART command.

This command is only valid if HP Instrument BASIC is installed.

DISPlay[1|2]:RESults — command/query

Select the data to be displayed in the specified trace.

Command Syntax: `DISPlay[1|2]:RESults <param>`

```
<param> ::= {SPECtrum|NORMalize|F1:F5|K1:K5|D1:D8}
```

```
Example Statements: Output 719;"display1:results spectrum"
                    Output 719;"Disp:Res Norm"
```

Query Syntax: `DISPlay[1|2]:RESults?`

Return Format: `<param>`

```
<param> ::= {SPEC|NORM|F1:F5|K1:K5|D1:D8}
```

Attribute Summary:

```
Preset state: SPEC (both displays)
Overlapped: yes
Pass control required: no
```

Description:

SPEC displays the results of the current measurement. NORM displays a normalized spectrum (SPEC/D8). F1 through F5 display the results of the corresponding trace math function. K1 through K5 display the amplitude of the corresponding trace math constants. D1 through D8 display the contents of the corresponding data registers.

DISPlay[1|2]:Y:SCALe:AUTO

command/query

Rescales and repositions the trace vertically to provide the best display of trace data.

Command Syntax: `DISPlay[1|2]:Y:SCALe:AUTO {ONCE|OFF|0}`

Example Statements:
```
Output 719;"DISP2:Y:SCAL:AUTO ONCE"
Output 719;"display2:y:scale:auto 0"
```

Query Syntax: `DISPlay[1|2]:Y:SCALe:AUTO?`

Return Format: +0

Attribute Summary:
```
Preset state: +0 (both displays)
Overlapped: yes
Pass control required: no
```

Description:

Send DISP:Y:SCAL:AUTO ONCE to initiate vertical autoscaling of the specified trace. The analyzer's autoscaling algorithm changes the values of DISP:Y:SCAL:MAX and DISP:Y:SCAL:PDIV to provide the best display of your data.

OFF has no effect on the analyzer. ON is not a valid option, because the analyzer does not support a continuous autoscaling mode.

DISPlay[1|2]:Y:SCALe:MAXimum

command/query

Sets the reference level in the specified display.

Command Syntax: `DISPlay[1|2]:Y:SCALe:MAXimum {<value>|<step>|<bound>}`

```
  <value> ::= <number>[<unit>]
 <number> ::= a real number (NRf data)
              limits:  -140.0:50.0
   <unit> ::= DBM|VRMS
   <step> ::= UP|DOWN
  <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Display1:Y:Scale:Maximum -15dBm"
                    Output 719;"DISP:Y:SCAL:MAX 10"
```

Query Syntax: `DISPlay[1|2]:Y:SCALe:MAXimum? [UNIT]`

Return Format: `<number>|{"<unit>"}`

```
 <number> ::= a real number (NR2 or NR3 data)
   <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: not defined
Overlapped: yes
Pass control required: no
```

Description:

The reference level determines the upper limit of the specified trace display area. The trace moves up or down in the display area when you change the reference level.

You can link the reference level to the current input range with the DISP:Y:SCAL:MAX:AUTO command.

DISPlay[1|2]:Y:SCALe:MAXimum:AUTO — command/query

Turns reference level tracking on and off.

Command Syntax: `DISPlay[1|2]:Y:SCALe:MAXimum:AUTO {OFF|0|ON|1}`

Example Statements:
```
Output 719;"disp:y:scal:max:auto 1"
Output 719;"Display1:Y:Scale:Maximum:Auto On"
```

Query Syntax: `DISPlay[1|2]:Y:SCALe:MAXimum:AUTO?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1 (both displays)
Overlapped: yes
Pass control required: no
```

Description:

If reference level tracking is enabled, the analyzer maintains the current offset between reference level (DISP:Y:SCAL:MAX) and input range (SENS:POW:RANG) whenever the range changes.

Note

Reference level never tracks input range changes while constants or data registers are displayed (DISP:RES Kx or DISP:RES Dx).

DISPlay[1|2]:Y:SCALe:PDIVision

command/query

Compresses or expands displayed data along its vertical axis.

Command Syntax: DISPlay[1|2]:Y:SCALe:PDIVision {<value>|<step>|<bound>}

```
  <value> ::= <number>[<unit>]
 <number> ::= a real number (NRf data)
              limits:  1.0E-3:50.0
   <unit> ::= DB|VRMS
   <step> ::= UP|DOWN
  <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"DISPLAY2:Y:SCALE:PDIVISION 2.5"
                    Output 719;"displ:y:scal:pdiv 40 mvrms"
```

Query Syntax: DISPlay[1|2]:Y:SCALe:PDIVision? [UNIT]

Return Format: <number>|{"<unit>"}

```
 <number> ::= a real number (NR2 or NR3 data)
   <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +1.00E+1 dBm (both displays)
Overlapped: yes
Pass control required: no
```

Description:

This command defines the height of each vertical division on the specified trace. When the trace display format is single (SCR:FORM SING) or front/back (SCR:FORM FBAC), the number of vertical divisions is 10. When the format is upper/lower (SCR:FORM ULOW), the number of vertical divisions is 5.

16

FORMat Subsystem

The FORMat subsystem contains a single command—FORM:DATA. The command determines which data type and data encoding will be used when large blocks of numeric data are transferred between the HP 3588A and a controller.

FORMat[:DATA] command/query

Specifies the data type and data encoding to be used during transfers of some block data.

Command Syntax: `FORMat[:DATA] {ASCii|REAL}[,{<number>|<bound>}]`

```
<number> ::= an integer (NRf data)
             limits:  3:64
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Form Asc,10"
                    Output 719;"FORMAT:DATA REAL"
```

Query Syntax: `FORMat[:DATA]?`

Return Format: `{ASC|REAL},<number>`

```
<number> ::= an integer (NR1 data)
             limits:  3:64
```

Attribute Summary:

```
Preset state: ASC,3
Overlapped: no
Pass control required: no
```

Description:

FORM:DATA only affects data transfers initiated by the following commands:

- CALC:DATA?
- DISP:LIM:LOW:REP?
- DISP:LIM:LOW:SEGM
- DISP:LIM:UPP:REP?
- DISP:LIM:UPP:SEGM
- PROG:SEL:NUMB
- TRAC:DATA

FORM:DATA ASC selects NRf data for transfers *to* the analyzer and NR3 data for transfers *from* the analyzer. Data encoding is ASCII. You control the number of significant digits in the returned numbers with the second parameter, which has a range of 3 through 12 when the first parameter is ASC.

FORM:DATA REAL selects definite or indefinite length block data for transfers *to* the analyzer but only definite length block data for transfers *from* the analyzer. Data encoding is binary (the binary floating-point format defined in the IEEE 754-1985 standard). The only allowed values for the second parameter are 32 and 64; it determines how many bits will be used for each number.

See "Data Encoding for Block Data" in chapter 4 for more information.

17

INITiate Subsystem

The INITiate subsystem contains two commands used to control initiation of the trigger system in TMSL instruments. However, neither of these commands affects the HP 3588A's trigger system because it differs slightly from the TMSL definition. The commands are included so you can write programs that work with the HP 3588A and with other TMSL instruments.

INITiate:CONTinuous

command/query

Sets the trigger system to a continuously initiated state.

Command Syntax: `INITiate:CONTinuous {ON|1}`

Example Statements:
```
Output 719;"initiate:continuous on"
Output 719;"Init:Cont 1"
```

Query Syntax: `INITiate:CONTinuous?`

Return Format: +1

Attribute Summary:
```
Preset state: +1
Overlapped: no
Pass control required: no
```

Description:

The HP 3588A's trigger system is always continuously initiated. As a result, this command has no effect on the analyzer state. It is included so you can write programs that work with this analyzer and with other TMSL instruments.

INITiate[:IMMediate] **command**

Forces the triggering system to exit the idle state.

Command Syntax: `INITiate[:IMMediate]`

Example Statements:
```
Output 719;"INIT"
Output 719;"initiate:immediate"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The HP 3588A's trigger system never enters the idle state. As a result, this command has no effect on the analyzer state. It is included so you can write programs that work with this analyzer and with other TMSL instruments.

Note

The program message *ABOR; INIT:IMM* is equivalent to the message *SENS:REST*. Both restart a measurement. However, SENS:REST is not supported by the TMSL standard.

18

INPut Subsystem

INPut is one of two subsystems that control the characteristics of the analyzer's input circuitry—the other is SENSe (under its POWer mnemonic). The commands in the INPut subsystem control input impedance and the input-protection relay.

INPut:IMPedance

command/query

Selects the impedance of the analyzer's input circuitry.

Command Syntax: `INPut:IMPedance (<value>|<bound>)`

```
      <value> ::= <number>[<unit>]
     <number> ::= a real number (NRf data)
                  limits:  50.0:1.0E6
                  discrete values: 50, 75, 1E6
       <unit> ::= OHM
      <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Input:Impedance 50 Ohm"
                    Output 719;"INP:IMP 1E6"
```

Query Syntax: `INPut:IMPedance? [UNIT]`

Return Format: `<number>|("<unit>")`

```
     <number> ::= a real number (NR2 or NR3 data)
       <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +5.00E+1 ohm
Overlapped: yes
Pass control required: no
```

Description:

When you select an input impedance of 1 megohm, dBm calculations use the reference impedance value you specify with the INP:IMP:REF command. Also, only the 0 dBm input range (referenced to 50 ohms) is available. (Input range is set with the SENS:POW:RANG command.)

Note

When you select an input impedance of 75 ohms, you *must use the 25 ohm adapter barrel* (supplied with the instrument) for accurate results. Insert the adapter between your test signal and the input connector.

INPut:IMPedance:REFerence — command/query

Selects the reference impedance to be used for dBm calculations when the input impedance is 1 megohm.

Command Syntax: `INPut:IMPedance:REFerence {<value>|<step>|<bound>}`

```
<value> ::= <number>[<unit>]
<number> ::= a real number (NRf data)
             limits:  1.0:1.0E6
<unit> ::= OHM
<step> ::= UP|DOWN
<bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"inp:imp:ref 600 ohm"
                    Output 719;"Input:Impedance:Reference 50"
```

Query Syntax: `INPut:IMPedance:REFerence? [UNIT]`

Return Format: `<number>|{"<unit>"}`

```
<number> ::= a real number (NR2 or NR3 data)
<unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +5.00E+1 ohm
Overlapped: yes
Pass control required: no
```

Description:

The value entered with this command is only used for dBm calculations when you select the 1 megohm input impedance (INP:IMP 1 MOHM). The dBm calculations determine trace amplitude values when the logarithmic magnitude coordinate system is selected (CALC:FORM MLOG).

INPut:TRIP:CLEar

command

Resets the analyzer's input-protection relay.

Command Syntax: `INPut:TRIP:CLEar`

```
Example Statements: Output 719;"INPUT:TRIP:CLEAR"
                    Output 719;"inp:trip:cle"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

The input-protection relay is tripped (opened) when the signal level at the input connector is significantly above the maximum input range. Bit 2 (Input Tripped) of the Questionable Power condition register tells you if the input-protection relay has been tripped.

19

MARKer Subsystem

The MARKer subsystem lets you control most of the analyzer's marker functions and read marker values. One marker function not controlled by this subsystem is limit testing. Commands that control limit testing and limit data are found in the DISPlay subsystem (under its LIMit mnemonic).

The MARKer mnemonic contains an optional trace specifier: [1|2]. To direct a command to trace A, omit the specifier or use 1. To direct a command to trace B, use 2. Commands that are not trace-specific—like MARK:STAT—ignore the specifier.

MARKer[1|2]:FUNCtion:FCOunt — command/query

Turns the frequency counter on and off.

Command Syntax: `MARKer[1|2]:FUNCtion:FCOunt {OFF|0|ON|1}`

Example Statements:
```
Output 719;"Mark:Func:Fco 0"
Output 719;"MARKER:FUNCTION:FCOUNT ON"
```

Query Syntax: `MARKer[1|2]:FUNCtion:FCOunt?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +0
Overlapped: yes
Pass control required: no
```

Description:

The frequency counter determines the frequency of the largest signal at the main marker position. You can move the main marker with MARK:X, MARK:POIN, MARK:MIN, or one of the MARK:MAX commands. You can read the counted frequency with the MARK:X:FCO query.

MARKer[1|2]:FUNCtion:NOISe — command/query

Turns the noise level function on and off.

Command Syntax: `MARKer[1|2]:FUNCtion:NOISe {OFF|0|ON|1}`

Example Statements:

```
Output 719;"marker:function:noise on"
Output 719;"Mark:Func:Nois Off"
```

Query Syntax: `MARKer[1|2]:FUNCtion:NOISe?`

Return Format: `+{0|1}`

Attribute Summary:

```
Preset state: +0
Overlapped: yes
Pass control required: no
```

Description:

The noise level marker determines the noise spectral density (normalized to a 1 Hz bandwidth) at the main marker position. You can move the main marker with MARK:X, MARK:POIN, MARK:MIN, or one of the MARK:MAX commands. You can read the noise spectral density with the MARK:Y:NOIS query.

Note

The noise level marker is not available for narrow band zoom measurements (SENS:FUNC 'POW:FFT').

MARKer[1|2]:MAXimum:GLOBal **command**

Moves the main marker to the highest peak on the specified trace.

Command Syntax: `MARKer[1|2]:MAXimum:GLOBal`

Example Statements:

```
Output 719;"MARK2:MAX:GLOB"
Output 719;"marker:maximum:global"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command moves the marker to the highest peak one time. Another command—MARK:MAX:TRAC—controls a marker function that automatically moves the marker to the highest peak *each time the trace is updated.*

MARKer[1|2]:MAXimum:LEFT — command

Moves the main marker one peak to the left of its current location on the specified trace.

Command Syntax: `MARKer[1|2]:MAXimum:LEFT`

Example Statements:

```
Output 719;"Marker1:Maximum:Left"
Output 719;"MARK2:MAX:LEFT"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

A peak is a local maximum on a trace. The slope of a trace is positive to the left of a peak and negative to the right. In addition, the slope on one side of a peak must not change for at least one vertical division (one-half division if SCR:FORM is ULOW).

This command only finds peaks that are at least one point to the left of the current marker location. If the peak search algorithm doesn't find a peak, the marker doesn't move. You can increase the number of peaks found by decreasing the value of DISP:Y:SCAL:PDIV.

MARKer[1|2]:MAXimum:RIGHt — command

Moves the main marker one peak to the right of its current location on the specified trace.

Command Syntax: `MARKer[1|2]:MAXimum:RIGHt`

Example Statements:

```
Output 719;"mark:max:righ"
Output 719;"Marker2:Maximum:Right"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

A peak is a local maximum on a trace. The slope of a trace is positive to the left of a peak and negative to the right. In addition, the slope on one side of a peak must not change for at least one vertical division (one-half division if SCR:FORM is ULOW).

This command only finds peaks that are at least one point to the right of the current marker location. If the peak search algorithm doesn't find a peak, the marker doesn't move. You can increase the number of peaks found by decreasing the value of DISP:Y:SCAL:PDIV.

MARKer[1|2]:MAXimum:TRACk **command/query**

Turns the peak tracking function on and off.

Command Syntax: `MARKer[1|2]:MAXimum:TRACk {OFF|0|ON|1}`

Example Statements:
```
Output 719;"MARKER:MAXIMUM:TRACK ON"
Output 719;"mark:max:trac 0"
```

Query Syntax: `MARKer[1|2]:MAXimum:TRACk?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +0
Overlapped: no
Pass control required: no
```

Description:

When peak tracking is enabled, the analyzer automatically positions the main marker on the largest peak of the active trace (SCR:ACT) each time the trace is updated. If you just want to move the marker to the highest peak *one time*, use the MARK:MAX:GLOB command.

MARKer[1|2]:MINimum:GLOBal

command

Moves the main marker to the lowest point on the specified trace.

Command Syntax: `MARKer[1|2]:MINimum:GLOBal`

Example Statements:

```
Output 719;"Mark1:Min:Glob"
Output 719;"MARKER2:MINIMUM:GLOBAL"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command moves the marker to the lowest peak one time. To move the marker to the *highest* peak one time, use the MARK:MAX:GLOB command.

MARKer[1|2]:OFFSet

command/query

Turns the offset marker on and off in the specified trace.

Command Syntax: `MARKer[1|2]:OFFSet {OFF|0|ON|1}`

Example Statements:
```
Output 719;"marker2:offset on"
Output 719;"Mark:Offs 1"
```

Query Syntax: `MARKer[1|2]:OFFSet?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +0 (both traces)
Overlapped: no
Pass control required: no
```

Description:

This command just controls the *display* of offset markers. You can always set and query an offset marker's position, even when it is not being displayed. Use MARK:OFFS:X, MARK:OFFS:Y, MARK:OFFS:DELT:X, and MARK:OFFS:DELT:Y to set and query an offset marker's position.

MARKer[1|2]:OFFSet:DELTa:X — command/query

Specifies the offset marker's x-axis position as an offset from the main marker's position.

Command Syntax: `MARKer[1|2]:OFFSet:DELTa:X {<value>|<bound>}`

```
<value>  ::= <number>[<unit>]
<number> ::= a real number (NRf data)
             limits:  -150.0E6:150.0E6
<unit>   ::= HZ|MHZ|S
<bound>  ::= MAX|MIN
```

```
Example Statements: Output 719;"MARK2:OFFS:DELT:X 10KHZ"
                    Output 719;"marker1:offset:delta:x 5e6"
```

Query Syntax: `MARKer[1|2]:OFFSet:DELTa:X? [UNIT]`

Return Format: `<number>|{"<unit>"}`

```
<number> ::= a real number (NR2 or NR3 data)
<unit>   ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +0.00E+0 Hz (both traces)
Overlapped: no
Pass control required: no
```

Description:

This command specifies the offset marker's x-axis position as an offset from the main marker's position. To specify the offset marker's absolute x-axis position, use the MARK:OFFS:X command.

MARKer[1|2]:OFFSet:DELTa:Y **command/query**

Specifies the offset marker's y-axis position as an offset from the main marker's position.

Command Syntax: `MARKer[1|2]:OFFSet:DELTa:Y {<value>|<bound>}`

```
  <value> ::= <number>[<unit>]
 <number> ::= a real number (NRf data)
              limits:  -50.0:50.0
   <unit> ::= DB|VRMS
  <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Marker1:Offset:Delta:Y 3dB"
                    Output 719;"MARK:OFFS:DELT:Y 7"
```

Query Syntax: `MARKer[1|2]:OFFSet:DELTa:Y?`

Return Format: `<number>|{"<unit>"}`

```
 <number> ::= a real number (NR2 or NR3 data)
   <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: not defined
Overlapped: no
Pass control required: no
```

Description:

This command specifies the offset marker's y-axis position as an offset from the main marker's position. To specify the offset marker's absolute y-axis position, use the MARK:OFFS:Y command.

MARKer[1|2]:OFFSet:X

command/query

Specifies the offset marker's x-axis position.

Command Syntax: `MARKer[1|2]:OFFSet:X {<value>|<step>|<bound>}`

```
   <value> ::= <number>[<unit>]
  <number> ::= a real number (NRf data)
               limits:  0.0:150.0E6
    <unit> ::= HZ|MHZ|S
    <step> ::= UP|DOWN
   <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"mark2:offs:x 50e6"
                    Output 719;"Marker2:Offset:X 400kHz"
```

Query Syntax: `MARKer[1|2]:OFFSet:X? [UNIT]`

Return Format: `<number>|{"<unit>"}`

```
  <number> ::= a real number (NR2 or NR3 data)
    <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +7.51E+7 Hz (both traces)
Overlapped: no
Pass control required: no
```

Description:

This command specifies the offset marker's absolute x-axis position. To specify the offset marker's x-axis position as an offset from the main marker's position, use the MARK:OFFS:DELT:X command.

MARKer[1|2]:OFFSet:Y

command/query

Specifies the offset marker's y-axis position.

Command Syntax:

```
MARKer[1|2]:OFFSet:Y {<value>|<bound>}

   <value> ::= <number>[<unit>]
  <number> ::= a real number (NRf data)
               limits:  -140.0:50.0
    <unit> ::= DBM|VRMS
   <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"MARKER1:OFFSET:Y 0.2 VRMS"
                    Output 719;"mark2:offs:y -10dbm"
```

Query Syntax:

```
MARKer[1|2]:OFFSet:Y? [UNIT]
```

Return Format:

```
<number>|{"<unit>"}

  <number> ::= a real number (NR2 or NR3 data)
    <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +0.00E+0 dBm (both traces)
Overlapped: no
Pass control required: no
```

Description:

This command specifies the offset marker's absolute y-axis position. To specify the offset marker's y-axis position as an offset from the main marker's position, use the MARK:OFFS:DELT:Y command.

MARKer[1|2]:POINt **command/query**

Moves the main marker to a particular display point.

Command Syntax: `MARKer[1|2]:POINt {<number>|<step>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:400
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Mark:Poin 0"
                    Output 719;"MARKER2:POINT 200"
```

Query Syntax: `MARKer[1|2]:POINt?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: +200 (both traces)
Overlapped: no
Pass control required: no
```

Description:

A trace is divided into 401 points along its x-axis. This command specifies the main marker's x-axis position by point number. To specify its x-axis position by frequency (or time), use the MARK:X command.

MARKer[1|2][:STATe] command/query

Enables the main markers or disables all markers and marker functions at once.

Command Syntax: `MARKer[1|2][:STATe] {OFF|0|ON|1}`

Example Statements:
```
Output 719;"marker:state off"
Output 719;"Mark 1"
```

Query Syntax: `MARKer[1|2][:STATe]?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: no
Pass control required: no
```

Description:

ON enables the display of the main markers.

OFF disables the display of main and offset markers for both traces, even if you append a trace specifier to the MARKer mnemonic. In addition, it disables the frequency counter (MARK:FUNC:FCO), the noise-level marker (MARK:FUNC:NOIS), and the peak tracking function (MARK:MAX:TRACK).

MARKer[1|2]:TO:SPAN — command

Resets the span to the frequency difference between the main and offset markers.

Command Syntax: `MARKer[1|2]:TO:SPAN`

Example Statements:
```
Output 719;"MARK2:TO:SPAN"
Output 719;"marker1:to:span"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

When swept spectrum measurements are selected (SENS:FUNC 'POW:SWEP'), this command resets the start frequency (SENS:FREQ:STAR) to the smaller of the two marker's frequencies. It resets the stop frequency (SENS:FREQ:STOP) to the larger of the two frequencies.

When narrow band zoom measurements are selected (SENS:FUNC 'POW:FFT'), this command resets the center frequency (SENS:FREQ:CENT) to one-half the sum of the main and offset marker frequencies. It resets the span (SENS:FREQ:SPAN) to the difference between these two frequencies. (If the difference doesn't exactly match one of the spans available for narrow band zoom measurements, the analyzer selects the next larger span.)

MARKer[1|2]:X

command/query

Specifies the main marker's x-axis position.

Command Syntax: MARKer[1|2]:X {<value>|<step>|<bound>}

```
    <value> ::= <number>[<unit>]
   <number> ::= a real number (NRf data)
                limits:  0.0:150.0E6
     <unit> ::= HZ|MHZ|S
     <step> ::= UP|DOWN
    <bound> ::= MAX|MIN
Example Statements: Output 719;"mark:x 98.1 mahz"
                    Output 719;"Marker:X 150ms"
```

Query Syntax: MARKer[1|2]:X? [UNIT]

Return Format: <number>|{"<unit>"}

```
   <number> ::= a real number (NR2 or NR3 data)
     <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +7.51E+7 Hz (both traces)
Overlapped: no
Pass control required: no
```

Description:

This command specifies the main marker's x-axis position by frequency (or time). To specify its x-axis position by display point number, use the MARK:POIN command.

MARKer[1|2]:X:FCOunt? query

Returns the last frequency measured by the frequency counter.

Query Syntax: `MARKer[1|2]:X:FCOunt? [UNIT]`

```
Example Statements: Output 719;"MARKER:X:FCOUNT?"
                    Output 719;"mark:x:fco?"
```

Return Format: `<number>|("<unit>")`

```
<number> ::= a real number (NR2 or NR3 data)
  <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: not defined
Overlapped: no
Pass control required: no
```

Description:

The frequency counter must be turned on before you can read a value with this query. Send MARK:FUNC:FCO ON to turn the frequency counter on.

MARKer[1|2]:Y? query

Returns the main marker's y-axis position.

Query Syntax: `MARKer[1|2]:Y? [UNIT]`

```
Example Statements: Output 719;"Mark:Y?"
                    Output 719;"MARKER1:Y?"
```

Return Format: `<number>|("<unit>")`

```
<number> ::= a real number (NR2 or NR3 data)
  <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: not defined
Overlapped: no
Pass control required: no
```

Description:

This query always returns the y-axis position of the main marker, even if the marker is not currently displayed on the analyzer's screen. The value returned tells you the amplitude of the specified trace at the marker's x-axis position (specified with MARK:X or MARK:POIN).

MARKer[1|2]:Y:NOISe? query

Returns the value last measured by the noise level marker.

Query Syntax: `MARKer[1|2]:Y:NOISe? [UNIT]`

```
Example Statements: Output 719;"marker:y:noise?"
                    Output 719;"Mark:Y:Nois?"
```

Return Format: `<number>|{"<unit>"}`

```
<number> ::= a real number (NR2 or NR3 data)
  <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: not defined
Overlapped: no
Pass control required: no
```

Description:

The noise level marker must be turned on before you can read a value with this query. Send MARK:FUNC:NOIS ON to turn on the noise-level marker.

20

MMEMory Subsystem

The MMEMory subsystem contains commands that control the analyzer's mass storage (disk) functions. Two of the mass storage devices are RAM-based disks—one using non-volatile RAM and the other using volatile RAM. The other mass storage device is an internal disk drive that uses 3.5-inch flexible disks. (The internal drive may be deleted on some instruments.)

Most MMEMory commands are directed to one of the disks with the following disk specifiers:

- NVRAM:—This specifies the non-volatile RAM disk.
- RAM:—This specifies the volatile RAM disk.
- INT:—This specifies the internal disk.

If you omit disk specifiers from MMEMory commands, the commands are automatically directed to the default disk. You select the default disk with the MMEM:MSI command.

MMEMory:COPY **command**

Copies the contents of one disk to another or one file to another.

Command Syntax: `MMEMory:COPY '<pathname>','<pathname>'`

```
    <pathname> ::= [<disk>][<filename>]
        <disk> ::= NVRAM:|RAM:|INT:
    <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                   underscore)
```

```
Example Statements: Output 719;"MMEM:COPY 'RAM:','INT:'"
                    Output 719;"mmemory:copy 'STATE1','RAM:STATE1'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

To copy a disk, just use a disk specifier for each <pathname>. To copy a file, use disk specifiers and filenames. If you just want to rename a file, use the MMEM:REN command.

MMEMory:DELete

command

Deletes one file or the contents of an entire disk.

Command Syntax: `MMEMory:DELete '[<disk>][<filename>]'`

```
<disk> ::= NVRAM:|RAM:|INT:
<filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
               underscore)
```

```
Example Statements: Output 719;"Mmemory:Delete 'RAM:'"
                    Output 719;"MMEM:DEL 'NVRAM:TRACE2'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

If you send the command with a disk specifier only, the contents of the entire disk are deleted.

MMEMory:GET:PROGram command

Loads an HP Instrument BASIC program into the analyzer from the specified disk.

Command Syntax:

```
MMEMory:GET:PROGram '[<disk>]<filename>'

      <disk> ::= NVRAM:|RAM:|INT:
  <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                 underscore); file must contain a program
```

```
Example Statements: Output 719;"mmem:get:prog 'MEAS_SEQ5'"
                    Output 719;"Mmemory:Get:Program 'RAM:ZERO_SPAN'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

You can load an HP Instrument BASIC program from disk either with this command or with the MMEM:LOAD:PROG command. However, only MMEM:LOAD:PROG is supported by the TMSL standard. You can load a program from your controller with the PROG:SEL:DEF command.

This command is only valid if the HP Instrument BASIC option is installed.

MMEMory:INITialize

command

Formats the specified disk.

Command Syntax: `MMEMory:INITialize '<disk>'[,{<number>|<bound>}]`

```
<disk> ::= NVRAM:|RAM:|INT:
<number> ::= an integer (NRf data)
             limits:  0:5
<bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"MMEMORY:INITIALIZE 'RAM:',5"
                    Output 719;"mmem:init 'INT:',1"
```

Attribute Summary:

```
Preset state: +0
Overlapped: no
Pass control required: no
```

Description:

The parameter you enter after the disk specifier is actually an encoded value that determines the disk's formatted capacity (in kilobytes):

Number	RAM Disk	NVRAM Disk	Internal Disk
0	64	63	640
1	640	▪	640
2	710	▪	710
3	788	▪	788
4	270	▪	—
5	640	▪	64

The interleave factor for the internal disk is always 1.

MMEMory:LOAD:LIMit:LOWer command

Loads a lower limit into the analyzer from the specified disk.

Command Syntax: `MMEMory:LOAD:LIMit:LOWer {A|B},'[<disk>]<filename>'`

```
      <disk> ::= NVRAM:|RAM:|INT:
  <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                 underscore); file must contain a limit or a trace
```

```
Example Statements: Output 719;"Mmem:Load:Lim:Low A,'INT:L_LIM2'"
                    Output 719;"MMEMORY:LOAD:LIMIT:LOWER B,'oldLim'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command loads the contents of a file into the lower limit register of the specified trace. The file must have been saved either with the MMEM:STOR:LIM:UPP, MMEM:STOR:LIM:LOW, or MMEM:STOR:TRAC command. Additional limit commands are available under DISP:LIM.

MMEMory:LOAD:LIMit:UPPer **command**

Loads an upper limit into the analyzer from the specified disk.

Command Syntax: `MMEMory:LOAD:LIMit:UPPer {A|B},'[<disk>]<filename>'`

```
<disk> ::= NVRAM:|RAM:|INT:
<filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
               underscore); file must contain a limit or a trace
```

```
Example Statements: Output 719;"mmemory:load:limit:upper b,'oldLim'"
                    Output 719;"Mmem:Load:Lim:Upp B,'INT:U_LIM2'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command loads the contents of a file into the upper limit register of the specified trace. The file must have been saved either with the MMEM:STOR:LIM:UPP, MMEM:STOR:LIM:LOW, or MMEM:STOR:TRAC command. Additional limit commands are available under DISP:LIM.

MMEMory:LOAD:MATH — command

Loads a complete set of math definitions into the analyzer from the specified disk.

Command Syntax: `MMEMory:LOAD:MATH '[<disk>]<filename>'`

```
       <disk> ::= NVRAM:|RAM:|INT:
   <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                  underscore); file must contain a set of math
                  definitions
```

```
Example Statements: Output 719;"MMEM:LOAD:MATH 'NVRAM:myMath'"
                    Output 719;"mmemory:load:math 'MATH1'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command uses the contents of a file to load all of the analyzer's function registers (F1 through F5) and constant registers (K1 through K5). The file must have been saved with the MMEM:STOR:MATH command.

MMEMory:LOAD:PROGram

command

Loads an HP Instrument BASIC program into the analyzer from the specified disk.

Command Syntax: `MMEMory:LOAD:PROGram '[<disk>]<filename>'`

```
        <disk> ::= NVRAM:|RAM:|INT:
    <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                   underscore); file must contain a program
```

```
Example Statements: Output 719;"Mmemory:Load:Program 'PROG2'"
                    Output 719;"MMEM:LOAD:PROG 'INT:MYPROG'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

You can load an HP Instrument BASIC program from disk either with this command or with the MMEM:GET:PROG command. However, only MMEM:LOAD:PROG is supported by the TMSL standard. You can load a program from your controller with the PROG:SEL:DEF command.

This command is only valid if the HP Instrument BASIC option is installed.

MMEMory:LOAD:STATe **command**

Loads an instrument state into the analyzer from the specified disk.

Command Syntax: `MMEMory:LOAD:STATe {1|MAX|MIN},'[<disk>]<filename>'`

```
       <disk> ::= NVRAM:|RAM:|INT:
   <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                  underscore); file must contain an instrument state
```

```
Example Statements: Output 719;"mmem:load:stat 1,'statel'"
                    Output 719;"Mmemory:Load:State 1,'STATE_2'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command uses the contents of a file to redefine the instrument state. The file must have been saved with the MMEM:STOR:STAT command.

MMEMory:LOAD:TRACe

command

Loads a trace into the analyzer from the specified disk.

Command Syntax: `MMEMory:LOAD:TRACe <data_reg>,'[<disk>]<filename>'`

```
     <data_reg> ::= (D1|D2|D3|D4|D5|D6|D7|D8)
         <disk> ::= NVRAM:|RAM:|INT:
     <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                    underscore); file must contain a trace
```

```
Example Statements: Output 719;"MMEMORY:LOAD:TRACE D6,'SPEC'"
                    Output 719;"mmem:load:trac d2,'INT:trace4'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command loads the contents of a file into one of the analyzer's eight data registers (D1 through D8). The file must have been saved with the MMEM:STOR:TRAC command. After loading the data register, you can display its contents with the DISP:RES command.

MMEMory:MSI

command/query

Selects a default disk for file and disk operations.

Command Syntax: `MMEMory:MSI '<disk>'`

```
<disk> ::= NVRAM:|RAM:|INT:
```

Example Statements:

```
Output 719;"Mmem:Msi 'INT:'"
Output 719;"MMEMORY:MSI 'NVRAM:'"
```

Query Syntax: `MMEMory:MSI?`

Return Format: `"<disk>"`

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

If you omit disk specifiers from MMEMory commands, the commands are automatically directed to the default disk. This command uses the following mnemonics to select the default disk:

- NVRAM:—This selects the non-volatile RAM disk.
- RAM:—This selects the volatile RAM disk.
- INT:—This selects the internal disk.

MMEMory:PACK

command

Increases the amount of usable space on the specified disk.

Command Syntax: `MMEMory:PACK ['<disk>']`

```
<disk> ::= NVRAM:|RAM:|INT:
```

```
Example Statements: Output 719;"mmemory:pack 'INT:'"
                    Output 719;"Mmem:Pack"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

When you delete files from a LIF-formatted disk, you sometimes leave spaces that are too small to be used for new files. This command recovers these unusable spaces.

MMEMory:REName **command**

Renames a file.

Command Syntax: `MMEMory:REName '<pathname>','<filename>'`

```
   <pathname> ::= [<disk>]<filename>
       <disk> ::= NVRAM:|RAM:|INT:
   <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                  underscore)
```

```
Example Statements: Output 719;"MMEM:REN 'INT:LIMIT','OLDLIMIT'"
                    Output 719;"mmemory:rename 'myprog','yourProg'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

Renaming only allows you to change a file's name on the current disk. It does not allow you to *move* a file by changing the file's name and disk specifier. To move a file, first copy it to another disk with the MMEM:COPY command, then delete it from the original disk with the MMEM:DEL command.

MMEMory:RESave:PROGram

command

Saves a program that has already been saved once.

Command Syntax: `MMEMory:RESave:PROGram '[<disk>]<filename>'`

```
<disk> ::= NVRAM:|RAM:|INT:
<filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
               underscore)
```

```
Example Statements: Output 719;"Mmemory:Resave:Program 'meas_seq5'"
                    Output 719;"MMEM:RES:PROG 'NVRAM:Program2'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command saves the current HP Instrument BASIC program to the specified disk. If the filename you specify matches the name of another file on the disk, the old file is overwritten. (Use MMEM:SAVE:PROG or MMEM:STOR:PROG if you want to ensure that the old file is not overwritten.)

This command is only valid when the HP Instrument BASIC option is installed.

MMEMory:SAVE:PROGram **command**

Saves a program for the first time.

Command Syntax: `MMEMory:SAVE:PROGram '[<disk>]<filename>'`

```
      <disk> ::= NVRAM:|RAM:|INT:
  <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                 underscore)
```

```
Example Statements: Output 719;"mmem:save:prog 'PROGRAM_1'"
                    Output 719;"Mmemory:Save:Program 'RAM:yourProg'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

You can use this command or the MMEM:STOR:PROG command to save the current HP Instrument BASIC program to the specified disk. (However, only MMEM:STOR:PROG is supported by the TMSL standard.) If the filename you specify matches the name of another file on the disk, the save is aborted to preserve the old file. (Use MMEM:RES:PROG if you want to overwrite the old file.)

This command is only valid when the HP Instrument BASIC option is installed.

MMEMory:STORe:LIMit:LOWer

command

Saves the specified lower limit to a disk.

Command Syntax: `MMEMory:STORe:LIMit:LOWer {A|B},'[<disk>]<filename>'`

```
      <disk> ::= NVRAM:|RAM:|INT:
  <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                 underscore)
```

Example Statements:

```
Output 719;"MMEMORY:STORE:LIMIT:LOWER B,'oldLim'"
Output 719;"MMEM:STORE:LIM:LOW A,'INT:L_LIM2'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The first parameter specifies which lower limit you are saving—the one from trace A or the one from trace B. If the filename you specify matches the name of another file on the disk, the old file is overwritten.

MMEMory:STORe:LIMit:UPPer — command

Saves the specified upper limit to a disk.

Command Syntax: `MMEMory:STORe:LIMit:UPPer {A|B},'[<disk>]<filename>'`

```
<disk> ::= NVRAM:|RAM:|INT:
<filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
               underscore)
```

Example Statements:
```
Output 719;"Mmem:Stor:Lim:Upp A,'RAM:uLim2'"
Output 719;"MMEMORY:STORE:LIMIT:UPPER A,'LIMIT'"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The first parameter specifies which upper limit you are saving—the one from trace A or the one from trace B. If the filename you specify matches the name of another file on the disk, the old file is overwritten.

MMEMory:STORe:MATH

command

Saves a complete set of math definitions to the specified disk.

Command Syntax: `MMEMory:STORe:MATH '[<disk>]<filename>'`

```
     <disk> ::= NVRAM:|RAM:|INT:
 <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                underscore)
```

```
Example Statements: Output 719;"mmemory:store:math 'MATH1'"
                    Output 719;"Mmem:Stor:Math 'NVRAM:myMath'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

A complete set of math definitions includes the current values in all of the function registers (F1 through F5) and all of the constant registers (K1 through K5). If the filename you specify with this command matches the name of another file on the disk, the old file is overwritten.

MMEMory:STORe:PROGram **command**

Saves a program for the first time.

Command Syntax: `MMEMory:STORe:PROGram '[<disk>]<filename>'`

```
        <disk> ::= NVRAM:|RAM:|INT:
    <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                   underscore)
```

```
Example Statements: Output 719;"MMEM:STOR:PROG 'PROG2'"
                    Output 719;"mmemory:store:program 'INT:myProg'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

You can use this command or MMEM:SAVE:PROG to save the current HP Instrument BASIC program to the specified disk. (However, only MMEM:STOR:PROG is supported by the TMSL standard.) If the filename you specify matches the name of another file on the disk, the save is aborted to preserve the old file. (Use MMEM:RES:PROG if you want to overwrite the old file.)

This command is only valid when the HP Instrument BASIC option is installed.

MMEMory:STORe:STATe — command

Saves the instrument state to the specified disk.

Command Syntax: `MMEMory:STORe:STATe {1|MAX|MIN},'[<disk>]<filename>'`

```
        <disk> ::= NVRAM:|RAM:|INT:
    <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                   underscore)
```

```
Example Statements: Output 719;"Mmemory:Store:State 1,'STATE_2'"
                    Output 719;"MMEM:STOR:STAT 1,'state1'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

If the filename you specify with this command matches the name of another file on the disk, the old file is overwritten.

MMEMory:STORe:TRACe **command**

Saves the specified trace to a disk.

Command Syntax: `MMEMory:STORe:TRACe {A|B},'[<disk>][<filename>]'`

```
        <disk> ::= NVRAM:|RAM:|INT:
    <filename> ::= 1 through 10 ASCII characters (use A:Z, a:z, 0:9, and
                   underscore)
```

```
Example Statements: Output 719;"mmem:stor:trac a,'INT:trace4'"
                    Output 719;"Mmemory:Store:Trace B,'SPEC'"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The first parameter specifies which trace you are saving—trace A or trace B. If the filename you specify matches the name of another file on the disk, the old file is overwritten.

21

PLOT Subsystem

The PLOT subsystem contains commands that control plotting parameters. It also contains commands that allow you to plot different portions of the analyzer's screen.

PLOT:ADDRess

command/query

Tells the analyzer which HP-IB address is assigned to your plotter.

Command Syntax: `PLOT:ADDRess {<number>|<step>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:30
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"PLOT:ADDRESS 0"
                    Output 719;"plot:addr 3"
```

Query Syntax: `PLOT:ADDRess?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

When you initiate a plot with one of the PLOT:DUMP commands, the analyzer expects to find a plotter at the HP-IB address specified with PLOT:ADDR. If there isn't a plotter at the specified address, the plot is automatically aborted.

PLOT:DUMP:ALL

command

Plots everything currently displayed on the analyzer's screen.

Command Syntax: `PLOT:DUMP:ALL`

Example Statements:
```
Output 719;"Plot:Dump:All"
Output 719;"PLOT:DUMP:ALL"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: yes
```

Description:

This command allows you to plot anything you can display with the SCR:CONT command. Everything on the screen is plotted *except* the softkey labels.

If you don't know how to pass control, see "Passing Control" in chapter 2.

PLOT:DUMP:GRATicule — command

Plots all displayed graticules.

Command Syntax: `PLOT:DUMP:GRATicule`

Example Statements:
```
Output 719;"plot:dump:graticule"
Output 719;"Plot:Dump:Grat"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: yes
```

Description:

Graticules are always plotted with solid lines, regardless of their appearance on the analyzer's screen.

If you don't know how to pass control, see "Passing Control" in chapter 2.

PLOT:DUMP:MARKer

command

Plots all displayed main markers and their coordinates.

Command Syntax: `PLOT:DUMP:MARKer`

Example Statements:
```
Output 719;"PLOT:DUMP:MARK"
Output 719;"plot:dump:marker"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: yes
```

Description:

Markers must be displayed (MARK:STAT ON) before they can be plotted. Markers are annotated with their x-axis and y-axis coordinates when you plot them with PLOT:DUMP:MARK.

If you don't know how to pass control, see "Passing Control" in chapter 2.

PLOT:DUMP:OFFSet:MARKer — command

Plots all displayed offset markers and their coordinates.

Command Syntax: `PLOT:DUMP:OFFSet:MARKer`

Example Statements:

```
Output 719;"Plot:Dump:Offset:Marker"
Output 719;"PLOT:DUMP:OFFS:MARK"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: yes
```

Description:

Offset markers must be displayed (MARK:OFFS ON) before they can be plotted. Offset markers are annotated with their x-axis and y-axis coordinates when you plot them with PLOT:DUMP:OFFS:MARK.

If you don't know how to pass control, see "Passing Control" in chapter 2.

PLOT:DUMP:TRACe

command

Plots all displayed traces.

Command Syntax: `PLOT:DUMP:TRACe`

Example Statements:

```
Output 719;"plot:dump:trac"
Output 719;"Plot:Dump:Trace"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: yes
```

Description:

When you use this command, displayed traces are plotted without graticules, annotation, or markers.

If you don't know how to pass control, see "Passing Control" in chapter 2.

PLOT:EJECt command/query

Turns the page-eject feature on and off for plotters that support such a feature.

Command Syntax: `PLOT:EJECt {OFF|0|ON|1}`

Example Statements:
```
Output 719;"PLOT:EJECT ON"
Output 719;"plot:ejec 0"
```

Query Syntax: `PLOT:EJECt?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: no
Pass control required: no
```

Description:

Check your plotter's documentation to be sure that it supports the requested page-eject state.

PLOT:LTYPe:TRACe[1|2] — command/query

Selects the line type for the specified trace.

Command Syntax: `PLOT:LTYPe:TRACe[1|2] {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  -4096:4096
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Plot:Ltyp:Trac1 -4096"
                    Output 719;"PLOT:LTYPE:TRACE2 1"
```

Query Syntax: `PLOT:LTYPe:TRACe[1|2]?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: -4096 (both traces)
Overlapped: no
Pass control required: no
```

Description:

The trace specifier determines whether you are selecting the line type for trace A or trace B. Omit the specifier or send 1 for trace A; send 2 for trace B. The <number> parameter is actually an encoded value. Encoded values for the most commonly used line types follow:

- Solid: -4096
- Dotted: 1
- Dashed: 2

Check your plotter's documentation to see if it supports additional line types.

PLOT:PEN:ALPHa command/query

Selects the pen to be used for plotting alpha characters.

Command Syntax: `PLOT:PEN:ALPHa {<number>|<step>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  1:16
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"plot:pen:alpha 6"
                    Output 719;"Plot:Pen:Alph 1"
```

Query Syntax: `PLOT:PEN:ALPHa?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: +4
Overlapped: no
Pass control required: no
```

Description:

The alpha pen is used to plot the instrument state and the disk catalog. (You can display these with the SCR:CONT command.)

PLOT:PEN:GRATicule

command/query

Selects the pen to be used for plotting graticules.

Command Syntax: `PLOT:PEN:GRATicule {<number>|<step>|<bound>}`

```
     <number> ::= an integer (NRf data)
                  limits:  1:16
       <step> ::= UP|DOWN
      <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"PLOT:PEN:GRAT 1"
                    Output 719;"plot:pen:graticule 2"
```

Query Syntax: `PLOT:PEN:GRATicule?`

Return Format: `<number>`

```
     <number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: +1
Overlapped: no
Pass control required: no
```

Description:

The graticule pen is used to plot trace graticules, the border around the instrument state, and the border around the disk catalog.

PLOT:PEN:INITialize

command

Returns plotter pen assignments to their default values.

Command Syntax: `PLOT:PEN:INITialize`

Example Statements:

```
Output 719;"Plot:Pen:Initialize"
Output 719;"PLOT:PEN:INIT"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

Here are the default pen values:

- PLOT:PEN:ALPH = 4
- PLOT:PEN:GRAT = 1
- PLOT:PEN:MARK1 = 5
- PLOT:PEN:MARK2 = 6
- PLOT:PEN:TRAC1 = 2
- PLOT:PEN:TRAC2 = 3

PLOT:PEN:MARKer[1|2]

command/query

Selects the pen used to plot markers for the specified trace.

Command Syntax: `PLOT:PEN:MARKer[1|2] {<number>|<step>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  1:16
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"plot:pen:markl 3"
                    Output 719;"Plot:Pen:Markerl 2"
```

Query Syntax: `PLOT:PEN:MARKer[1|2]?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: +5 (MARK1), +6 (MARK2)
Overlapped: no
Pass control required: no
```

Description:

The marker pen is used to plot markers and limit lines. The trace specifier you send with this command determines whether you are selecting the pen number for trace A markers or trace B markers. Omit the specifier or send 1 for trace A; send 2 for trace B.

PLOT:PEN:TRACe[1|2] command/query

Selects the pen used to plot the specified trace.

Command Syntax:

```
PLOT:PEN:TRACe[1|2] {<number>|<step>|<bound>}

     <number> ::= an integer (NRf data)
                  limits:  1:16
       <step> ::= UP|DOWN
      <bound> ::= MAX|MIN

Example Statements: Output 719;"plot:pen:trace2 10"
                    Output 719;"Plot:Pen:Trac2 1"
```

Query Syntax:

```
PLOT:PEN:TRACe[1|2]?
```

Return Format:

```
<number>

     <number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: +2 (TRAC1), +3 (TRAC2)
Overlapped: no
Pass control required: no
```

Description:

The trace pen is used to plot traces and all of the following trace-specific annotation:

- Trace title.
- Marker readout.
- X-axis annotation.
- Y-axis annotation.
- Limit test results.

The trace specifier you send with this command determines whether you are selecting the pen number for trace A or trace B. Omit the specifier or send 1 for trace A; send 2 for trace B.

PLOT:SPEed

command/query

Specifies the plotting speed that will be requested by the analyzer.

Command Syntax: `PLOT:SPEed {<number>|<step>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  1:100
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"PLOT:SPE 50"
                    Output 719;"plot:speed 10"
```

Query Syntax: `PLOT:SPEed?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: +50
Overlapped: no
Pass control required: no
```

Description:

You request a plot speed in units of cm/s. Check your plotter's documentation to be sure that it supports the requested plotting speed.

22

PRINt Subsystem

The PRINt subsystem contains two commands. One command tells the analyzer where to send print data; the other prints the contents of the analyzer's screen.

PRINt:ADDRess

command/query

Tells the analyzer which HP-IB address is assigned to your printer.

Command Syntax:

```
PRINt:ADDRess {<number>|<step>|<bound>}

    <number> ::= an integer (NRf data)
                 limits:  0:30
      <step> ::= UP|DOWN
     <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Print:Address 1"
                    Output 719;"PRIN:ADDR 14"
```

Query Syntax: `PRINt:ADDRess?`

Return Format: `<number>`

```
    <number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

When you initiate a print operation with the PRIN:DUMP:ALL command, the analyzer expects to find a printer at the HP-IB address specified with PRIN:ADDR. If there isn't a printer at the specified address, the print operation is automatically aborted.

PRINt:DUMP:ALL

command

Prints everything currently displayed on the analyzer's screen.

Command Syntax: `PRINt:DUMP:ALL`

Example Statements:

```
Output 719;"prin:dump:all"
Output 719;"Print:Dump:All"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: yes
```

Description:

This command allows you to print anything you can display with the SCR:CONT command. Everything on the screen is printed *except* the softkey labels.

Note

Print information is sent as a bit-mapped graphic, so your printer must accept raster dumps.

23

PROGram Subsystem

The commands in the PROGram subsystem are only available when the HP Instrument BASIC option is installed. They allow you to interact with the program currently loaded in the analyzer.

All of the commands in this subsystem are grouped under the mnemonic *SELected*. Since SELected is an implied mnemonic, you can omit it from all PROGram commands. See "Implied Mnemonics" in chapter 3 for more information.

PROGram[:SELected]:DEFine command/query

Loads an HP Instrument BASIC program into the analyzer from an external controller. (Use the query form to save.)

Command Syntax: `PROGram[:SELected]:DEFine <block>`

```
          <block> ::= #<byte>[<length_bytes>]<data_bytes>
           <byte> ::= one ASCII-encoded byte specifying the number of length
                      bytes to follow
   <length_bytes> ::= ASCII-encoded bytes specifying the number of data
                      bytes to follow
     <data_bytes> ::= the bytes that define a program
```

```
Example Statements: Output 719;"PROGRAM:SELECTED:DEFINE?"
                    Output 719;"prog:def?"
```

Query Syntax: `PROGram[:SELected]:DEFine?`

Return Format: `<block>`

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command transfers a program between the analyzer and your controller. This allows you to develop a program on your controller and then load it into the analyzer when it's done.

When you transfer a program to the analyzer, you can use either the definite or the indefinite length block syntax. When the analyzer returns the program to your controller, it always uses the definite length block syntax. See "Block Data" in chapter 4 for more information.

PROGram[:SELected]:DELete[:SELected] command

Deletes the current HP Instrument BASIC program.

Command Syntax: `PROGram[:SELected]:DELete[:SELected]`

Example Statements:

```
Output 719;"program:selected:delete:selected"
Output 719;"Prog:Del"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

In addition to deleting the current program, this command deletes all of the programs variables—both those in COM and those not in COM.

PROGram[:SELected]:MALLocate — command/query

Allocates stack space for HP Instrument BASIC programs.

Command Syntax: `PROGram[:SELected]:MALLocate {<number>|<bound>|DEFault}`

<number> ::= an integer (NRf data)
limits: 1200 bytes:3 megabytes

<bound> ::= MAX|MIN

Example Statements:

```
Output 719;"PROG:SEL:MALL DEF"
Output 719;"program:selected:mallocate 8192"
```

Query Syntax: `PROGram[:SELected]:MALLocate?`

Return Format: <number>

<number> ::= an integer (NR1 data)

Attribute Summary:

Preset state: not affected by Preset
Overlapped: no
Pass control required: no

Description:

Stack space is the portion of memory used for temporary storage of program variables (excluding those variables stored in COM). It provides the programs "working space."

Note

If you encounter the message "ERROR 2 Memory overflow." while your program is running, you need to allocate more stack space.

PROGram[:SELected]:NUMBer — command/query

Loads a new value for the specified numeric variable.

Command Syntax: `PROGram[:SELected]:NUMBer '<variable>',<block>`

```
<variable> ::= name of a numeric variable
```

When data is ASCII-encoded, (FORM ASC) <block> takes the following form:

```
   <block> ::= <number>[,<number>]...
  <number> ::= a real number (NRf data)
               limits:  -9.9E37:9.9E37
```

When data is binary-encoded, (FORM REAL) <block> takes the following form:

```
        <block> ::= #<byte>[<length_bytes>]<number>[<number>]...
         <byte> ::= one ASCII-encoded byte specifying the number of length
                    bytes to follow
<length_bytes> ::= ASCII-encoded bytes specifying the number of
                    data bytes to follow
       <number> ::= a real number (32- or 64-bit binary floating point)
                    limits:  -9.9E37:9.9E37
```

```
Example Statements: Output 719;"Program:Number 'Address',19"
                    Output 719;"PROG:NUMB 'Scode',7"
```

Query Syntax: `PROGram[:SELected]:NUMBer? '<variable>'`

Return Format: `<block>`

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

When you load an array with this command, values in the <block> parameter are loaded into the 1st through nth elements of the array (where *n* is the number of values in the block).

PROGram[:SELected]:STATe

command/query

Selects the state of the current HP Instrument BASIC program.

Command Syntax: `PROGram[:SELected]:STATe <param>`

`<param> ::= {STOP|PAUSe|RUN|CONTinue}`

Example Statements:
```
Output 719;"prog:sel:stat paus"
Output 719;"Program:State Continue"
```

Query Syntax: `PROGram[:SELected]:STATe?`

Return Format: `STOP|PAUS|RUN`

Attribute Summary:
```
Preset state: STOP
Overlapped: no
Pass control required: no
```

Description:

The analyzer generates an error message if you send RUN or CONT while a program is running. It also generates an error if you send CONT while a program is stopped.

PROGram[:SELected]:STRing — command/query

Loads a new value for the specified string variable.

Command Syntax: `PROGram[:SELected]:STRing '<variable>','<string>'`

```
      <variable> ::= name of a string variable
        <string> ::= 0 through 255 ASCII characters
Example Statements: Output 719;"PROGRAM:STRING 'A$','Done.'"
                    Output 719;"prog:str 'Message$','Measuring.'"
```

Query Syntax: `PROGram[:SELected]:STRing? '<variable>'`

Return Format: `"<string>"`

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

Use this command to load string variables. Use the PROG:SEL:NUMB command to load numeric variables.

24

SCReen Subsystem

SCReen is one of two subsystems that control the analyzer's presentation of data on its front-panel display. The other subsystem is DISPlay (excluding the commands grouped under its LIMit mnemonic).

SCReen:ACTive

command/query

Selects the active trace.

Command Syntax: `SCReen:ACTive {A|B}`

Example Statements:

```
Output 719;"Scr:Act A"
Output 719;"SCREEN:ACTIVE B"
```

Query Syntax: `SCReen:ACTive?`

Return Format: `A|B`

Attribute Summary:

```
Preset state: A
Overlapped: yes
Pass control required: no
```

Description:

Only the active trace is displayed on the analyzer's screen when SCR:FORM is SING.

SCReen:ANNotation

command/query

Enables and disables the display of frequency information on the analyzer's screen.

Command Syntax: `SCReen:ANNotation {OFF|0|ON|1}`

Example Statements:
```
Output 719;"screen:annotation off"
Output 719;"Scr:Ann 0"
```

Query Syntax: `SCReen:ANNotation?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: yes
Pass control required: no
```

Description:

When SCR:ANN is OFF, frequency information is not displayed on the analyzer's screen and it is not printed or plotted. However, all frequency information is still available via HP-IB.

SCReen:CONTents

command/query

Specifies what will be displayed on the analyzer's screen.

Command Syntax: `SCReen:CONTents {TRACe|STATe|MMEMory}`

Example Statements:

```
Output 719;"SCR:CONT STAT"
Output 719;"screen:contents trace"
```

Query Syntax: `SCReen:CONTents?`

Return Format: `TRAC|STAT|MMEM`

Attribute Summary:

```
Preset state: TRAC
Overlapped: yes
Pass control required: no
```

Description:

Send TRAC to display any trace data. (You can then choose the trace data you want to display with the DISP:RES command.) Send STAT to display the current state of the most important measurement setup parameters. Send MMEM to display the contents of the analyzer's default disk. (You select the default disk with the MMEM:MSI command.)

SCReen:FORMat

command/query

Selects a format for displaying trace data.

Command Syntax: `SCReen:FORMat {SINGle|ULOWer|FBACk}`

```
Example Statements: Output 719;"Screen:Format Single"
                    Output 719;"SCR:FORM ULOW"
```

Query Syntax: `SCReen:FORMat?`

Return Format: `SING|ULOW|FBAC`

Attribute Summary:

```
Preset state: SING
Overlapped: yes
Pass control required: no
```

Description:

When you select SING, the analyzer uses the entire trace display area for the active trace (SCR:ACT). When you select ULOW, the analyzer uses the upper half of the trace display area for trace A and the lower half for trace B. When you select FBAC, the analyzer uses the entire trace display area, but overlays the two traces in that area.

SCReen:GRATicule **command/query**

Turns the trace graticules on and off.

Command Syntax: `SCReen:GRATicule {OFF|0|ON|1}`

```
Example Statements: Output 719;"scr:grat on"
                    Output 719;"Screen:Graticule Off"
```

Query Syntax: `SCReen:GRATicule?`

Return Format: `+{0|1}`

Attribute Summary:

```
Preset state: +1
Overlapped: yes
Pass control required: no
```

Description:

When the graticules are turned off they are not displayed on the analyzer's screen and they are not plotted or printed.

SCReen[:STATe]

command/query

Turns the analyzer's screen on and off.

Command Syntax: `SCReen[:STATe] {OFF|0|ON|1}`

Example Statements:
```
Output 719;"SCREEN:STATE OFF"
Output 719;"scr 1"
```

Query Syntax: `SCReen[:STATe]?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: yes
Pass control required: no
```

Description:

When the screen is turned off, the message "DISPLAY BLANKING ON" replaces all other information. Only this message is plotted or printed.

25

[SENSe] Subsystem

Commands in the SENSe subsystem determine how measurement data will be acquired. Because SENSe is an implied mnemonic, you can omit from all SENSe commands. See "Implied Mnemonics" in chapter 3 for more information.

[SENSe:]BANDwidth:NOISe? query

Returns the noise equivalent bandwidth for the current measurement.

Query Syntax: `[SENSe:]BANDwidth:NOISe? [UNIT]`

```
Example Statements: Output 719;"SENSE:BANDWIDTH:NOISE?"
                    Output 719;"band:nois?"
```

Return Format: `<number>|{"Hz"}`

`<number> ::= a real number (NR2 or NR3 data)`

Attribute Summary:
```
Preset state: +1.81E+4 Hz
Overlapped: no
Pass control required: no
```

Description:

To create a power spectral density trace, you can divide each point of a linear magnitude spectrum (DISP:RES SPEC and CALC:FORM MLIN) by the square root of the noise equivalent bandwidth. You can transfer a linear magnitude spectrum to your controller with CALC:DATA.

[SENSe:]BANDwidth:NOISe:CORRection? query

Returns the noise correction factor for the current measurement.

Query Syntax: `[SENSe:]BANDwidth:NOISe:CORRection? [UNIT]`

```
Example Statements: Output 719;"SENSE:BANDWIDTH:NOISE:CORRECTION?"
                    Output 719;"band:nois:corr?"
```

Return Format: `<number>|{"dB"}`

`<number> ::= a real number (NR2 or NR3 data)`

Attribute Summary:

```
Preset state: +4.26E+1 dB
Overlapped: no
Pass control required: no
```

Description:

To create a power spectral density trace, you can subtract the noise correction factor from each point of a logarithmic magnitude spectrum (DISP:RES SPEC and CALC:FORM MLOG). You can transfer a logarithmic magnitude spectrum to your controller with CALC:DATA.

[SENSe:]BANDwidth[:RESolution] — command/query

Selects a value for the resolution bandwidth filter.

Command Syntax:

```
[SENSe:]BANDwidth[:RESolution] {<value>|<step>|<bound>}

  <value> ::= <number>[<unit>]
 <number> ::= a real number (NRf data)
              limits:  1.1:17000.0
              discrete values:  1.1, 2.3, 4.5, 9.1, 18, 36, 73,
              150, 290, 580, 1200, 2300, 4600, 9100, 17000
   <unit> ::= HZ|MHZ
   <step> ::= UP|DOWN
  <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Band Down"
                    Output 719;"SENSE:BANDWIDTH:RESOLUTION 4.6KHZ"
```

Query Syntax:

```
[SENSe:]BANDwidth[:RESolution]? [UNIT]
```

Return Format:

```
<number>|{"<unit>"}

 <number> ::= a real number (NR2 or NR3 data)
   <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +1.7E+4 Hz
Overlapped: yes
Pass control required: no
```

Description:

The resolution bandwidth filter determines the frequency resolution of swept spectrum measurements (SENS:FUNC 'POW:SWEP').

Note

When bandwidth coupling is on (SENS:BAND:RES:AUTO ON), changes in frequency span (SENS:FREQ:SPAN) can force automatic changes in the resolution bandwidth setting.

[SENSe:]BANDwidth[:RESolution]:AUTO — command/query

Turns bandwidth coupling on and off.

Command Syntax: `[SENSe:]BANDwidth[:RESolution]:AUTO {OFF|0|ON|1|ONCE}`

Example Statements:
```
Output 719;"bandwidth:resolution:auto once"
Output 719;"Sens:Band:Auto 1"
```

Query Syntax: `[SENSe:]BANDwidth[:RESolution]:AUTO?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: yes
Pass control required: no
```

Description:

When bandwidth coupling is on, the analyzer couples (creates dependencies among) the following parameters:

- Frequency span, set with SENS:FREQ:SPAN.
- Resolution bandwidth (RBW), set with SENS:BAND:RES.
- Video bandwidth (VBW), set with SENS:BAND:VID.
- Sweep time, set with SENS:SWE:TIME.

Here is the order of dependency among coupled parameters (from least dependent to most):

Span —> RBW —> VBW —> Sweep time

If you change a particular parameter when bandwidth coupling is on, the analyzer changes the more dependent parameters automatically. For example, if you change RBW, the analyzer changes VIDEO BANDWIDTH and sweep time. Be sure to set the least dependent parameters first when bandwidth coupling is on. See the help text or your *HP 3588A Operating Manual* for more information on bandwidth coupling.

When you send SENS:BAND:RES:AUTO ONCE, you restore the optimum relationships between the coupled parameters. This provides the best compromise between frequency resolution and speed for most measurements.

[SENSe:]BANDwidth[:RESolution]:FFT? query

Returns the measurement resolution value for the current narrow band zoom measurement.

Query Syntax: `[SENSe:]BANDwidth[:RESolution]:FFT? [UNIT]`

```
Example Statements: Output 719;"SENSE:BANDWIDTH:RESOLUTION:FFT?"
                    Output 719;"band:fft?"
```

Return Format: `<number>|{"Hz"}`

```
          <number> ::= a real number (NR2 or NR3 data)
```

Attribute Summary:

```
Preset state: +3.59E+2 Hz
Overlapped: no
Pass control required: no
```

Description:

The measurement resolution value is dependent on span (SENS:FREQ:SPAN) and zoom type (SENS:WIND:TYPE). It is only valid for narrow band zoom measurements (SENS:FUNC 'POW:FFT'). It is analogous to the resolution bandwidth value (SENS:BAND:RES) used for swept spectrum measurements.

[SENSe:]BANDwidth:VIDeo **command/query**

Selects a value for the video bandwidth filter.

Command Syntax:

```
[SENSe:]BANDwidth:VIDeo {<value>|<step>|<bound>}

  <value> ::= <number>[<unit>]
 <number> ::= a real number (NRf data)
              limits:  0.019:26248.0
   <unit> ::= HZ|MHZ
   <step> ::= UP|DOWN
  <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"BAND:VID UP"
                    Output 719;"bandwidth:video 2000"
```

Query Syntax:

```
[SENSe:]BANDwidth:VIDeo? [UNIT]
```

Return Format:

```
<number>|{"<unit>"}

 <number> ::= a real number (NR2 or NR3 data)
   <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +2.62E+4 Hz
Overlapped: yes
Pass control required: no
```

Description:

The video filter has a trace smoothing function for swept spectrum measurements (SENS:FUNC 'POW:SWEP'). You increase the amount of smoothing by decreasing the bandwidth of the filter.

Note

When bandwidth coupling is on (SENS:BAND:RES:AUTO ON), changes in frequency span (SENS:FREQ:SPAN) or resolution bandwidth (SENS:BAND:RES) can force automatic changes in the video bandwidth setting.

Use the SENS:BAND:VID:STAT command to turn the video filter on and off.

[SENSe:]BANDwidth:VIDeo:STATe — command/query

Turns video filtering on and off.

Command Syntax: `[SENSe:]BANDwidth:VIDeo:STATe {OFF|0|ON|1}`

Example Statements:
```
Output 719;"Sense:Bandwidth:Video:State Off"
Output 719;"BAND:VID:STAT 1"
```

Query Syntax: `[SENSe:]BANDwidth:VIDeo:STATe?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +0
Overlapped: yes
Pass control required: no
```

Description:

This command turns the video filter on and off. To set the filter's bandwidth, use the SENS:BAND:VID command.

[SENSe:]DETector[:FUNCtion]

command/query

Turns the peak detector on and off.

Command Syntax: `[SENSe:]DETector[:FUNCtion] {POSitive|SAMPle}`

Example Statements:
```
Output 719;"det pos"
Output 719;"Sense:Detector:Function Positive"
```

Query Syntax: `[SENSe:]DETector[:FUNCtion]?`

Return Format: `POS|SAMP`

Attribute Summary:
```
Preset state: POS
Overlapped: yes
Pass control required: no
```

Description:

POSitive turns the peak detector on. SAMPle turns the peak detector off. See the help text or your *HP 3588A Operating Manual* for more information about the peak detector.

Note

The peak detector should be on for all swept spectrum measurements. It should be off for scalar network measurements (made with the analyzer's source).

[SENSe:]DETector:STIMe — command/query

Specifies the sampling time for manually swept measurements.

Command Syntax: `[SENSe:]DETector:STIMe {<value>|<step>|<bound>}`

```
 <value> ::= <number>[S]
<number> ::= a real number (NRf data)
             limits:  4.0E-6:4295.0
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"DETECTOR:STIME 4.0E-6 S"
                    Output 719;"det:stim up"
```

Query Syntax: `[SENSe:]DETector:STIMe? [UNIT]`

Return Format: `<number>|{"s"}`

```
<number> ::= a real number (NR2 or NR3 data)
```

Attribute Summary:

```
Preset state: +4.00E-6 s
Overlapped: yes
Pass control required: no
```

Description:

Sample time specifies how long the analyzer should measure at a single frequency during manual sweeps (SENS:SWE:MODE MAN). It does not specify the time *between* measurements, which can be significantly longer than the sample time.

[SENSe:]FREQuency:CENTer

command/query

Specifies the center frequency for the current measurement.

Command Syntax:

```
[SENSe:]FREQuency:CENTer {<value>|<step>|<bound>}

 <value> ::= <number>[<unit>]
<number> ::= a real number (NRf data)
             limits:  0.0:150.0E6
  <unit> ::= HZ|MHZ
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Sens:Freq:Cent 20e6 Hz"
                    Output 719;"SENSE:FREQUENCY:CENTER 98.1 MAHZ"
```

Query Syntax:

```
[SENSe:]FREQuency:CENTer? [UNIT]
```

Return Format:

```
<number>|{"<unit>"}

<number> ::= a real number (NR2 or NR3 data)
  <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +7.505E+7 Hz
Overlapped: yes
Pass control required: no
```

Description:

SENS:FREQ:CENT and SENS:FREQ:SPAN work together to define the band of frequencies you want to analyze. The current value of one parameter is held constant when you change the value of the other.

Note

When SENS:FREQ:SPAN is set to 0, the analyzer acts as a fixed-tuned receiver and SENS:FREQ:CENT tunes the receiver to the desired frequency.

[SENSe:]FREQuency:CENTer:TRACk — command/query

Turns the signal tracking function on and off.

Command Syntax: `[SENSe:]FREQuency:CENTer:TRACk {OFF|0|ON|1}`

```
Example Statements: Output 719;"frequency:center:track off"
                    Output 719;"Sens:Freq:Cent:Trac 0"
```

Query Syntax: `[SENSe:]FREQuency:CENTer:TRACk?`

Return Format: `+{0|1}`

Attribute Summary:

```
Preset state: +0
Overlapped: yes
Pass control required: no
```

Description:

When signal tracking is turned on, the analyzer automatically tracks a drifting signal. It works by adjusting the value of SENS:FREQ:CENT to keep the largest signal centered in the current frequency span (SENS:FREQ:SPAN).

[SENSe:]FREQuency:MANual

command/query

Specifies the measurement frequency during manually swept measurements.

Command Syntax:

```
[SENSe:]FREQuency:MANual {<value>|<step>|<bound>}

  <value> ::= <number>[<unit>]
 <number> ::= a real number (NRf data)
              limits:  0.0:150.0E6
   <unit> ::= HZ|MHZ
   <step> ::= UP|DOWN
  <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"FREQ:MAN 114980000"
                    Output 719;"frequency:manual 490khz"
```

Query Syntax:

```
[SENSe:]FREQuency:MANual? [UNIT]
```

Return Format:

```
<number>|{"<unit>"}

 <number> ::= a real number (NR2 or NR3 data)
   <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +7.505E+7 Hz
Overlapped: yes
Pass control required: no
```

Description:

The manual frequency value is only used if manually sweeping is enabled (SENS:SWE:MODE MAN). The range of values you can specify for manual frequency is limited by the current values of SENS:FREQ:STAR and SENS:FREQ:STOP.

[SENSe:]FREQuency:SPAN — command/query

Specifies the frequency span for the current measurement.

Command Syntax:

```
[SENSe:]FREQuency:SPAN {<value>|<step>|<bound>}

 <value> ::= <number>[<unit>]
<number> ::= a real number (NRf data)
             limits:  0.0:150.0E6
  <unit> ::= HZ|MHZ
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

Example Statements:

```
Output 719;"Sense:Frequency:Span 40 kHz"
Output 719;"FREQ:SPAN DOWN"
```

Query Syntax:

```
[SENSe:]FREQuency:SPAN? [UNIT]
```

Return Format:

```
<number>|{"<unit>"}

<number> ::= a real number (NR2 or NR3 data)
  <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +1.499E+8 Hz
Overlapped: yes
Pass control required: no
```

Description:

SENS:FREQ:SPAN and SENS:FREQ:CENT work together to define the band of frequencies you want to analyze. The current value of one parameter is held constant when you change the value of the other.

Note

When SENS:FREQ:SPAN is set to 0, the analyzer acts as a fixed-tuned receiver and SENS:FREQ:CENT tunes the receiver to the desired frequency. This measurement setup is called *zero span*.

During swept spectrum measurements (SENS:FUNC 'POW:SWEP'), the frequency span can be 0 Hz or it can be within the range of 10 Hz to 150 MHz. During narrow band zoom measurements (SENS:FUNC 'POW:FFT), the frequency span is limited to 16 discrete values ranging from 1.22 Hz to 40 kHz. The values are derived from the following formula:

$$40{,}000 / 2^n$$

where n has integer values ranging from 0 to 15.

When bandwidth coupling is on (SENS:BAND:RES:AUTO ON) changes in span can force automatic changes in the following parameters:

- Resolution bandwidth, set with SENS:BAND:RES
- Video bandwidth, set with SENS:BAND:VID
- Sweep time, set with SENS:SWE:TIME

[SENSe:]FREQuency:SPAN:FULL command

Sets the analyzer to the widest frequency span available for the current measurement type.

Command Syntax: `[SENSe:]FREQuency:SPAN:FULL`

Example Statements:
```
Output 719;"freq:span:full"
Output 719;"Sense:Frequency:Span:Full"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

During swept spectrum measurements (SENS:FUNC 'POW:SWEP'), this command sets SENS:FREQ:SPAN to 150 MHz and SENS:FREQ:CENT to 75 MHz. During narrow band zoom measurements (SENS:FUNC 'POW:FFT'), this command sets SENS:FREQ:SPAN to 40 kHz and leaves SENS:FREQ:CENT unchanged.

[SENSe:]FREQuency:STARt — command/query

Specifies the start frequency for the current measurement.

Command Syntax: `[SENSe:]FREQuency:STARt {<value>|<step>|<bound>}`

```
   <value> ::= <number>[<unit>]
  <number> ::= a real number (NRf data)
               limits:  0.0:150.0E6
    <unit> ::= HZ|MHZ
    <step> ::= UP|DOWN
   <bound> ::= MAX|MIN
```

Example Statements:
```
Output 719;"FREQUENCY:START 10"
Output 719;"sens:freq:star 100khz"
```

Query Syntax: `[SENSe:]FREQuency:STARt? [UNIT]`

Return Format: `<number>|{"<unit>"}`

```
  <number> ::= a real number (NR2 or NR3 data)
    <unit> ::= unit that applies to returned number
```

Attribute Summary:
```
Preset state: +1.00E+5 Hz
Overlapped: yes
Pass control required: no
```

Description:

SENS:FREQ:STAR and SENS:FREQ:STOP work together to define the band of frequencies you want to analyze. SENS:FREQ:STAR defines the band's lower limit, SENS:FREQ:STOP defines its upper limit. Here's how they work together for each of the two measurement types:

- Swept spectrum (SENS:FUNC 'POW:SWEP')—The current value of one parameter is held constant when you change the value of the other.
- Narrow band zoom (SENS:FUNC 'POW:FFT')—The offset between the two parameters is held constant when you change the value of either.

Note

During swept spectrum measurements, the span changes when you change the start frequency. This can cause changes in other parameters if bandwidth coupling is on. See SENS:BAND:RES:AUTO for more information.

[SENSe:]FREQuency:STEP

command/query

Specifies the step size to be used for changing frequency parameters.

Command Syntax: `[SENSe:]FREQuency:STEP {<value>|<step>|<bound>}`

```
      <value> ::= <number>[<unit>]
     <number> ::= a real number (NRf data)
                  limits:  -150.0E6:150.0E6
       <unit> ::= HZ|MHZ
       <step> ::= UP|DOWN
      <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Freq:Step 60 Hz"
                    Output 719;"SENSE:FREQUENCY:STEP 1E6"
```

Query Syntax: `[SENSe:]FREQuency:STEP? [UNIT]`

Return Format: `<number>|{"<unit>"}`

```
     <number> ::= a real number (NR2 or NR3 data)
       <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +1.00E+3 Hz
Overlapped: yes
Pass control required: no
```

Description:

Step size determines the frequency change that results when you send UP or DOWN with any of the following commands:

- SENS:FREQ:CENT
- SENS:FREQ:STAR
- SENS:FREQ:STOP
- SENS:FREQ:MAN

The step size you specify with SENS:FREQ:STEP is only used for the listed commands when SENS:FREQ:STEP:STAT is MAN.

[SENSe:]FREQuency:STEP:STATe — command/query

Specifies whether the step size for frequency parameters is determined by the analyzer or by the value of SENS:FREQ:STEP.

Command Syntax: `[SENSe:]FREQuency:STEP:STATe {AUTO|MANual}`

Example Statements:
```
Output 719;"sense:frequency:step:state auto"
Output 719;"Freq:Step:Stat Man"
```

Query Syntax: `[SENSe:]FREQuency:STEP:STATe?`

Return Format: `AUTO|MAN`

Attribute Summary:
```
Preset state: AUTO
Overlapped: yes
Pass control required: no
```

Description:

SENS:FREQ:STEP:STAT allows you to enable either the analyzer-determined (AUTO) or the user-determined (MAN) step size for the following frequency commands:

- SENS:FREQ:CENT
- SENS:FREQ:STAR
- SENS:FREQ:STOP
- SENS:FREQ:MAN

Step size determines the change that results when you send UP or DOWN with a command. Use SENS:FREQ:STEP to define the user-determined step size for the listed commands.

[SENSe:]FREQuency:STOP

command/query

Specifies the stop frequency for the current measurement.

Command Syntax:

```
[SENSe:]FREQuency:STOP {<value>|<step>|<bound>}

  <value> ::= <number>[<unit>]
 <number> ::= a real number (NRf data)
              limits:  0.0:150.0E6
   <unit> ::= HZ|MHZ
   <step> ::= UP|DOWN
  <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"FREQ:STOP 150E+6 HZ"
                    Output 719;"frequency:stop 480khz"
```

Query Syntax:

```
[SENSe:]FREQuency:STOP? [UNIT]
```

Return Format:

```
<number>|{"<unit>"}

 <number> ::= a real number (NR2 or NR3 data)
   <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +1.5E+8 Hz
Overlapped: yes
Pass control required: no
```

Description:

SENS:FREQ:STOP and SENS:FREQ:STAR work together to define the band of frequencies you want to analyze. SENS:FREQ:STOP defines the band's upper limit, SENS:FREQ:STAR defines its lower limit. Here's how they work together for each of the two measurement types:

- Swept spectrum (SENS:FUNC 'POW:SWEP')—The current value of one parameter is held constant when you change the value of the other.
- Narrow band zoom (SENS:FUNC 'POW:FFT')—The offset between the two parameters is held constant when you change the value of either.

Note

During swept spectrum measurements, the span changes when you change the stop frequency. This can cause changes in other parameters if bandwidth coupling is on. See SENS:BAND:RES:AUTO for more information.

[SENSe:]FUNCtion

command/query

Selects one of the analyzer's two major measurement types.

Command Syntax: `[SENSe:]FUNCtion '{POWer:FFT|POWer:SWEPt}'`

Example Statements:
```
Output 719;"Function 'Power:Fft'"
Output 719;"SENS:FUNC 'POW:SWEP'"
```

Query Syntax: `[SENSe:]FUNCtion?`

Return Format: `"{POWER:FFT|POWER:SWEPT}"`

Attribute Summary:
```
Preset state: "POWER:SWEPT"
Overlapped: yes
Pass control required: no
```

Description:

When you switch between swept spectrum (SENS:FUNC 'POW:SWEP') and narrow band zoom (SENS:FUNC 'POW:FFT') measurements, the state of many other parameters can change. As a result, you should select the measurement type near the beginning of any program sequence that defines the instrument state.

See the help text or your *HP 3588A Operating Manual* for information on the analyzer's two measurement types.

[SENSe:]FUNCtion:POWer:FFT — command

Selects the narrow band zoom measurement as the current measurement type.

Command Syntax: `[SENSe:]FUNCtion:POWer:FFT`

Example Statements:
```
Output 719;"func:pow:fft"
Output 719;"Sense:Function:Power:Fft"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

When you switch between narrow band zoom (SENS:FUNC:POW:FFT) and swept spectrum (SENS:FUNC:POW:SWEP) measurements, the state of many other parameters can change. As a result, you should select the measurement type near the beginning of any program sequence that defines the instrument state.

Note

Since this command has no query form, you must use the SENS:FUNC query to determine which measurement type is currently selected. The query returns 'POW:FFT' when narrow band zoom is selected.

See the help text or your *HP 3588A Operating Manual* for information on the analyzer's two measurement types.

[SENSe:]FUNCtion:POWer:SWEPt

command

Selects the swept spectrum measurement as the current measurement type.

Command Syntax: `[SENSe:]FUNCtion:POWer:SWEPt`

```
Example Statements: Output 719;"FUNCTION:POWER:SWEPT"
                    Output 719;"func:pow:swep"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

When you switch between swept spectrum (SENS:FUNC:POW:SWEP) and narrow band zoom (SENS:FUNC:POW:FFT) measurements, the state of many other parameters can change. As a result, you should select the measurement type near the beginning of any program sequence that defines the instrument state.

Note

Since this command has no query form, you must use the SENS:FUNC query to determine which measurement type is currently selected. The query returns 'POW:SWEP' when swept spectrum is selected.

See the help text or your *HP 3588A Operating Manual* for information on the analyzer's two measurement types.

[SENSe:]POWer:RANGe **command/query**

Selects the sensitivity of the analyzer's input circuitry.

Command Syntax: `[SENSe:]POWer:RANGe {<value>|<step>|<bound>}`

```
   <value> ::= <number>[<unit>]
  <number> ::= a real number (NRf data)
               limits:  -20.0:20.0
    <unit> ::= DBM|VRMS
    <step> ::= UP|DOWN
   <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Pow:Rang Up"
                    Output 719;"SENSE:POWER:RANGE 0 DBM"
```

Query Syntax: `[SENSe:]POWer:RANGe? [UNIT]`

Return Format: `<number>|{"<unit>"}`

```
  <number> ::= a real number (NR2 or NR3 data)
    <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: not defined
Overlapped: yes
Pass control required: no
```

Description:

The range setting determines the maximum ac signal level that can be applied to the analyzer's input connector without overdriving the input circuitry. The following table shows the range settings available for each of the analyzer's three input impedances (INP:IMP).

50Ω	75Ω	1 MΩ
20 dBm (2.24 Vrms)	21.76 dBm (3.35 Vrms)	
10 dBm (707 mVrms)	11.76 dBm (1.06 Vrms)	
0 dBm (223 mVrms)	1.76 dBm (335 mVrms)	0 dBm * (223 mVrms) (– 13 dBV)
– 10 dBm (70.7 mVrms)	– 8.23 dBm (106 mVrms)	
– 20 dBm (22.4 mVrms)	– 18.23 dBm (33.5 mVrms)	

* Referenced to 50Ω

You can set the range manually with SENS:POW:RANG, or you can let the analyzer select the range automatically with SENS:POW:RANG:AUTO. If you set the range manually, two bits in the Questionable Power condition register help you decide when to change the range setting:

- Bit 0 (Input Overloaded) is set to 1 when *any signal between 0 and 150 MHz* exceeds the current input range.
- Bit 4 (ADC Overloaded) is set to 1 when *any signal in the current span* is overloading the analyzer's analog-to-digital converter.

Note

If reference level tracking is on (DISP:Y:SCAL:MAX:AUTO ON), the reference level setting (DISP:Y:SCAL:MAX) changes automatically when the range changes.

[SENSe:]POWer:RANGe:AUTO — command/query

Automatically selects the best range for the current input signal.

Command Syntax: `[SENSe:]POWer:RANGe:AUTO {OFF|0|ON|1|ONCE}`

Example Statements:
```
Output 719;"sense:power:range:auto on"
Output 719;"Pow:Rang:Auto Once"
```

Query Syntax: `[SENSe:]POWer:RANGe:AUTO?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: yes
Pass control required: no
```

Description:

Send SENS:POW:RANG:AUTO ONCE to execute the analyzer's autoranging algorithm *one time* for the current input signal. Send SENS:POW:RANG:AUTO ON to enable the algorithm to *continuously* monitor the input signal and adjust the range.

The autoranging algorithm selects the best input range for the current input signal. The algorithm selects a less sensitive range if the signal level is large enough to overdrive the input circuitry. It selects a more sensitive range if the signal level is small enough to compromise dynamic range. (See SENS:POW:RANG for more information on the available ranges.)

[SENSe:]POWer:RANGe:LDIStortion

command/query

Enables and disables the analyzer's low distortion mode.

Command Syntax: `[SENSe:]POWer:RANGe:LDIStortion {OFF|0|ON|1}`

Example Statements:

```
Output 719;"SENS:POW:RANG:LDIS 0"
Output 719;"power:range:ldistortion 1"
```

Query Syntax: `[SENSe:]POWer:RANGe:LDIStortion?`

Return Format: `+{0|1}`

Attribute Summary:

```
Preset state: +0
Overlapped: yes
Pass control required: no
```

Description:

Low-distortion mode reduces the distortion contribution of the HP 3588A's analog input circuitry. If you are measuring signal components that are within 10 dB of the analyzer's published distortion specification, this mode may improve your measurement results.

Refer to the help text or the *HP 3588A Operating Manual* for more information on low-distortion mode.

[SENSe:]RESTart

command

Immediately stops the current measurement and starts a new one.

Command Syntax: `[SENSe:]RESTart`

```
Example Statements: Output 719;"Sense:Restart"
                    Output 719;"REST"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

Although SENS:REST allows you to abort the current measurement and start a new one it is not supported by the TMSL standard. The program message ABOR;INIT:IMM aborts the current measurement and starts a new one for *any* TMSL instrument.

Both SENS:REST and ABOR;INIT:IMM serve a special synchronizing function. When you send either of these commands to restart a measurement, the analyzer's No Pending Operation (NPO) flag is not set to 1 until the measurement is complete. The two commands that test the state of this flag—*WAI and *OPC—allow you to hold off subsequent actions until the measurement is complete. See "Synchronization" in chapter 2 for more information on the NPO flag.

Note

When video averaging is enabled (AVER:TYPE RMS and AVER:STAT ON), the NPO flag is not set to 1 until *n* measurements have been combined into one trace. You specify the value of *n* with the AVER:COUN command.

[SENSe:]SWEep:MODE

command/query

Specifies whether sweeping is done automatically or manually.

Command Syntax: `[SENSe:]SWEep:MODE {AUTO|MANual}`

Example Statements:
```
Output 719;"swe:mode man"
Output 719;"Sense:Sweep:Mode Auto"
```

Query Syntax: `[SENSe:]SWEep:MODE?`

Return Format: `AUTO|MAN`

Attribute Summary:
```
Preset state: AUTO
Overlapped: yes
Pass control required: no
```

Description:

When you select automatic sweeping (AUTO), the analyzer sweeps from SENS:FREQ:STAR to SENS:FREQ:STOP at a speed determined by SENS:SWE:TIME. When you select manual sweeping (MAN), the analyzer measures at a single frequency—the manual frequency (SENS:FREQ:MAN). The analyzer samples the manual frequency for an amount of time determined by SENS:DET:STIM.

The setting of SENS:SWE:MODE is ignored for narrow band zoom measurements (SENS:FUNC 'POW:FFT').

[SENSe:]SWEep:TIME

command/query

Specifies the sweep time for automatically swept measurements.

Command Syntax: `[SENSe:]SWEep:TIME {<value>|<step>|<bound>}`

```
  <value> ::= <number>[S]
 <number> ::= a real number (NRf data)
              limits:  1.0E-3:72.0E3
   <step> ::= UP|DOWN
  <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"SWEEP:TIME DOWN"
                    Output 719;"swe:time 400 ms"
```

Query Syntax: `[SENSe:]SWEep:TIME? [UNIT]`

Return Format: `<number>|{"s"}`

```
::= a real number (NR2 or NR3 data)
```

Attribute Summary:

```
Preset state: +2.608E-1 s
Overlapped: yes
Pass control required: no
```

Description:

This command specifies how long the analyzer takes to sweep from SENS:FREQ:STAR to SENS:FREQ:STOP. If you select a sweep time that is too short to provide calibrated measurement results, bit 1 (Uncal Oversweep) of the Questionable Power condition register is set to 1.

When bandwidth coupling is on (SENS:BAND:RES:AUTO ON), changes in any of the following parameters can force automatic changes in the sweep time setting:

- Frequency span, set with SENS:FREQ:SPAN.
- Resolution bandwidth, set with SENS:BAND:RES.
- Video bandwidth, set with SENS:BAND:VID.

[SENSe:]WINDow[:TYPE] command/query

Determines the frequency resolution of narrow band zoom measurements.

Command Syntax: `[SENSe:]WINDow[:TYPE] {HANNing|FLATtop}`

Example Statements:
```
Output 719;"Wind Flat"
Output 719;"SENSE:WINDOW:TYPE HANNING"
```

Query Syntax: `[SENSe:]WINDow[:TYPE]?`

Return Format: `HANN|FLAT`

Attribute Summary:
```
Preset state: FLAT
Overlapped: yes
Pass control required: no
```

Description:

HANN selects the Hanning window, which provides better frequency resolution for narrow band zoom measurements (SENS:FUNC 'POW:FFT'). FLAT selects the Flat Top window, which provides better amplitude accuracy. Sending HANN is equivalent to pressing the [HI RES ZOOM] softkey on the analyzer's front panel. Sending FLAT is equivalent to pressing the [HI ACCRCY ZOOM] softkey.

26

SOURce Subsystem

Commands in the SOURce subsystem control the analyzer's source (tracking generator).

SOURce:OUTPut:IMPedance — command/query

Selects the impedance of the analyzer's source.

Command Syntax: `SOURce:OUTPut:IMPedance {<value>|<bound>}`

```
  <value> ::= <number>[<unit>]
 <number> ::= an integer (NRf data)
              limits:  50~75
              discrete values:  50, 75
   <unit> ::= OHM|MEGOHM
  <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"source:output:impedance 50"
                    Output 719;"Sour:Outp:Imp 75 Ohm"
```

Query Syntax: `SOURce:OUTPut:IMPedance? [UNIT]`

Return Format: `<number>|{"<unit>"}`

```
 <number> ::= an integer (NR1 data)
   <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +50 ohm
Overlapped: yes
Pass control required: no
```

Description:

You can select the source's output impedance explicitly with this command, or you can couple it to the input impedance with the SOUR:OUPT:IMP:MODE command.

Note

When you select an output impedance of 75 ohms, you *must use the 25 ohm adapter barrel* (supplied with the instrument) for accurate results. Insert the adapter between the analyzer's source connector and your test device.

SOURce:OUTPut:IMPedance:MODE

command/query

Turns impedance coupling on and off.

Command Syntax: `SOURce:OUTPut:IMPedance:MODE {OFF|0|ON|1}`

Example Statements:
```
Output 719;"SOUR:OUTP:IMP:MODE 1"
Output 719;"source:output:impedance:mode on"
```

Query Syntax: `SOURce:OUTPut:IMPedance:MODE?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: yes
Pass control required: no
```

Description:

When impedance coupling is turned on, the output impedance (SOUR:OUTP:IMP) tracks changes in the input impedance (INP:IMP) with one exception: If you switch from a 50 or 75 ohm input impedance to a 1 megohm input impedance, the output impedance retains its current setting.

Note If you set the output impedance explicitly with the SOUR:OUTP:IMP command, impedance coupling is automatically turned off.

SOURce:OUTPut:PROTection:CLEar **command**

Resets the analyzer's source-protection relay.

Command Syntax: `SOURce:OUTPut:PROTection:CLEar`

```
Example Statements: Output 719;"Source:Output:Protection:Clear"
                    Output 719;"SOUR:OUTP:PROT:CLE"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

The source-protection relay is tripped (opened) when the signal level at the analyzer's source connector is significantly above the maximum source amplitude or when excessive dc voltage is present. Bit 3 (Source Tripped) of the Questionable Power condition register tells you if the source-protection relay has been tripped.

SOURce:OUTPut[:STATe]

command/query

Turns the analyzer's source on and off.

Command Syntax: `SOURce:OUTPut[:STATe] {OFF|0|ON|1}`

Example Statements:

```
Output 719;"sour:outp 0"
Output 719;"Source:Output:State On"
```

Query Syntax: `SOURce:OUTPut[:STATe]?`

Return Format: `+{0|1}`

Attribute Summary:

```
Preset state: +0
Overlapped: yes
Pass control required: no
```

Description:

When the source is off, the output amplitude is approximately -100 dBm.

SOURce:POWer[:LEVel][:IMMediate][:AMPLitude] command/query

Specifies the output amplitude of the analyzer's source.

Command Syntax: `SOURce:POWer[:LEVel][:IMMediate][:AMPLitude] <param>`

```
<param> ::= {<number>[<unit>]}|<step>|<bound>
<number> ::= a real number (NRf data)
             limits:  -61.7:10.0
  <unit> ::= DBM|VRMS
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"SOUR:POW:LEV:IMM:AMPL 0 DBM"
                    Output 719;"source:power .2 vrms"
```

Query Syntax: `SOURce:POWer[:LEVel][:IMMediate][:AMPLitude]? [UNIT]`

Return Format: `<number>|{"<unit>"}`

```
<number> ::= a real number (NR2 or NR3 data)
  <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: -1.00E+1 dBm
Overlapped: yes
Pass control required: no
```

Description:

The smallest possible increment between source amplitude values is 0.1 dBm. You can select the increment with the SOUR:POW:LEV:IMM:AMPL:STEP command.

SOURce:POWer[:LEVel][:IMMediate][:AMPLitude]:STEP — command/query

Specifies the step size to be used for changing the source amplitude.

Command Syntax: `SOURce:POWer[:LEVel][:IMMediate][:AMPLitude]:STEP <param>`

```
    <param> ::= {<number>[<unit>]}|<step>|<bound>
   <number> ::= a real number (NRf data)
                limits:   -61.7:10.0
     <unit> ::= DB|VRMS
     <step> ::= UP|DOWN
    <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Sour:Pow:Lev:Imm:Ampl:Step 0.1"
                    Output 719;"SOURCE:POWER:IMMEDIATE:STEP 400MVRMS"
```

Query Syntax: `SOURce:POWer[:LEVel][:IMMediate][:AMPLitude]:STEP? [UNIT]`

Return Format: `<number>|("<unit>")`

```
   <number> ::= a real number (NR2 or NR3 data)
     <unit> ::= unit that applies to returned number
```

Attribute Summary:

```
Preset state: +1.00E-1 dB
Overlapped: yes
Pass control required: no
```

Description:

Step size determines the amplitude change that results when you send UP or DOWN with the SOUR:POW:LEV:IMM:AMPL command.

27

STATus Subsystem

The STATus subsystem provide access to most of the HP 3588A's status reporting structures (register sets). Some of the common commands (described in chapter 8) provide access to the other register sets.

Most of the commands in this subsystem are used to set bits in registers; most of the queries are used to read registers. Decimal weights are assigned to bits according to the following formula:

$$\text{weight} = 2^n$$

where *n* is the bit number (with acceptable values of 0 through 14).

To set a single register bit to 1, send the decimal weight of that bit with the command that writes the register. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. Queries that read registers always return the sum of the decimal weights of all bits that are currently set to 1.

See chapter 5 for more information about the analyzer's register sets.

STATus:DEVice:CONDition? query

Reads the Device State condition register.

Query Syntax: `STATus:DEVice:CONDition?`

Example Statements:

```
Output 719;"status:device:condition?"
Output 719;"Stat:Dev:Cond?"
```

Return Format: <number>

<number> ::= an integer (NR1 data)

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Device State condition register. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Device State Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of condition registers in register sets.

STATus:DEVice:ENABle

command/query

Sets bits in the Device State enable register.

Command Syntax: `STATus:DEVice:ENABle {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"STAT:DEV:ENAB 0"
                    Output 719;"status:device:enable 4"
```

Query Syntax: `STATus:DEVice:ENABle?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Device State enable register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Device State Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of enable registers in register sets.

STATus:DEVice[:EVENt]? query

Reads and clears the Device State event register.

Query Syntax: `STATus:DEVice[:EVENt]?`

```
Example Statements: Output 719;"Status:Device:Event?"
                    Output 719;"STAT:DEV?"
```

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Device State event register. (The decimal weight of a bit is 2^n, where *n* is the bit number.)

Note The Device State event register is automatically cleared after it is read by this query.

See "Device State Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of event registers in register sets.

STATus:DEVice:NTRansition **command/query**

Sets bits in the Device State negative transition register.

Command Syntax: `STATus:DEVice:NTRansition {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits: 0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"stat:dev:ntr 2"
                    Output 719;"Status:Device:Ntransition 7"
```

Query Syntax: `STATus:DEVice:NTRansition?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Device State negative transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Device State Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of negative transition registers in register sets.

STATus:DEVice:PTRansition — command/query

Sets bits in the Device State positive transition register.

Command Syntax: `STATus:DEVice:PTRansition {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"STATUS:DEVICE:PTRANSITION 4"
                    Output 719;"stat:dev:ptr 6"
```

Query Syntax: `STATus:DEVice:PTRansition?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Device State positive transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Device State Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of positive transition registers in register sets.

STATus:OPERation:CONDition? query

Reads the Standard Operation condition register.

Query Syntax: `STATus:OPERation:CONDition?`

Example Statements:
```
Output 719;"Stat:Oper:Cond?"
Output 719;"STATUS:OPERATION:CONDITION?"
```

Return Format: <number>

<number> ::= an integer (NR1 data)

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Standard Operation condition register. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Standard Operation Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of condition registers in register sets.

STATus:OPERation:ENABle — command/query

Sets bits in the Standard Operation enable register.

Command Syntax: `STATus:OPERation:ENABle {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"status:operation:enable 16384"
                    Output 719;"Stat:Oper:Enab 64"
```

Query Syntax: `STATus:OPERation:ENABle?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Standard Operation enable register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Standard Operation Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of enable registers in register sets.

STATus:OPERation[:EVENt]? query

Reads and clears the Standard Operation event register.

Query Syntax: `STATus:OPERation[:EVENt]?`

Example Statements:
```
Output 719;"STAT:OPER?"
Output 719;"status:operation:event?"
```

Return Format: `<number>`

`<number> ::= an integer (NR1 data)`

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Standard Operation event register. (The decimal weight of a bit is 2^n, where n is the bit number.)

Note

The Standard Operation event register is automatically cleared after it is read by this query.

See "Standard Operation Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of event registers in register sets.

STATus:OPERation:NTRansition — command/query

Sets bits in the Standard Operation negative transition register.

Command Syntax:

```
STATus:OPERation:NTRansition {<number>|<bound>}

          <number> ::= an integer (NRf data)
                       limits:  0:32767
           <bound> ::= MAX|MIN

Example Statements: Output 719;"Status:Operation:Ntransition 1"
                    Output 719;"STAT:OPER:NTR 65535"
```

Query Syntax: `STATus:OPERation:NTRansition?`

Return Format: `<number>`

```
          <number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Standard Operation negative transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Standard Operation Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of negative transition registers in register sets.

STATus:OPERation:PTRansition **command/query**

Sets bits in the Standard Operation positive transition register.

Command Syntax: `STATus:OPERation:PTRansition {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767
 <bound> ::= MAX|MIN
```

Example Statements:
```
Output 719;"stat:oper:ptr 16"
Output 719;"Status:Operation:Ptransition 0"
```

Query Syntax: `STATus:OPERation:PTRansition?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:
```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Standard Operation positive transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where *n* is the bit number.)

See "Standard Operation Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of positive transition registers in register sets.

STATus:PRESet — command

Sets bits in most enable and transition registers to their default state.

Command Syntax: `STATus:PRESet`

Example Statements:
```
Output 719;"Stat:Pres"
Output 719;"STATUS:PRESET"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

STAT:PRES has the following effect on the Limit Fail, Questionable Frequency, and Questionable Power register sets:

- Sets all enable register bits to 1.
- Sets all positive transition register bits to 1.
- Sets all negative transition register bits to 0.

It has the following effect on the Device State, Questionable Data, and Standard Operation register sets:

- Sets all enable register bits to 0.
- Sets all positive transition register bits to 1.
- Sets all negative transition register bits to 0.

STAT:PRES also sets all bits in the User Defined enable register to 0. It has no effect on any other register.

STATus:QUEStionable:CONDition? query

Reads the Questionable Data condition register.

Query Syntax: `STATus:QUEStionable:CONDition?`

Example Statements:
```
Output 719;"status:questionable:condition?"
Output 719;"Stat:Ques:Cond?"
```

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Questionable Data condition register. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Questionable Data Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of condition registers in register sets.

STATus:QUEStionable:ENABle — command/query

Sets bits in the Questionable Data enable register.

Command Syntax: `STATus:QUEStionable:ENABle {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits: 0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"STAT:QUES:ENAB 40"
                    Output 719;"status:questionable:enable 0"
```

Query Syntax: `STATus:QUEStionable:ENABle?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Questionable Data enable register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where *n* is the bit number.)

See "Questionable Data Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of enable registers in register sets.

STATus:QUEStionable[:EVENt]?

query

Reads and clears the Questionable Data event register.

Query Syntax: STATus:QUEStionable[:EVENt]?

Example Statements: Output 719;"Status:Questionable?"
Output 719;"STAT:QUES:EVEN?"

Return Format: <number>

<number> ::= an integer (NR1 data)

Attribute Summary: Preset state: not applicable
Overlapped: no
Pass control required: no

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Questionable Data event register. (The decimal weight of a bit is 2^n, where *n* is the bit number.)

Note

The Questionable Data event register is automatically cleared after it is read by this query.

See "Questionable Data Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of event registers in register sets.

STATus:QUEStionable:FREQuency:CONDition? query

Reads the Questionable Frequency condition register.

Query Syntax: `STATus:QUEStionable:FREQuency:CONDition?`

Example Statements:
```
Output 719;"stat:ques:freq:cond?"
Output 719;"Status:Questionable:Frequency:Condition?"
```

Return Format: `<number>`

`<number> ::= an integer (NR1 data)`

Attribute Summary:
```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Questionable Frequency condition register. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Questionable Frequency Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of condition registers in register sets.

STATus:QUEStionable:FREQuency:ENABle — command/query

Sets bits in the Questionable Frequency enable register.

Command Syntax: `STATus:QUEStionable:FREQuency:ENABle {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"STATUS:QUESTIONABLE:FREQUENCY:ENABLE 4"
                    Output 719;"stat:ques:freq:enab 5"
```

Query Syntax: `STATus:QUEStionable:FREQuency:ENABle?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Questionable Frequency enable register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where *n* is the bit number.)

See "Questionable Frequency Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of enable registers in register sets.

STATus:QUEStionable:FREQuency[:EVENt]? query

Reads and clears the Questionable Frequency event register.

Query Syntax: `STATus:QUEStionable:FREQuency[:EVENt]?`

```
Example Statements: Output 719;"Stat:Ques:Freq:Even?"
                    Output 719;"STATUS:QUESTIONABLE:FREQUENCY?"
```

Return Format: `<number>`

`<number> ::= an integer (NR1 data)`

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Questionable Frequency event register. (The decimal weight of a bit is 2^n, where n is the bit number.)

Note

The Questionable Frequency event register is automatically cleared after it is read by this query.

See "Questionable Frequency Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of event registers in register sets.

STATus:QUEStionable:FREQuency:NTRansition command/query

Sets bits in the Questionable Frequency negative transition register.

Command Syntax: `STATus:QUEStionable:FREQuency:NTRansition <param>`

```
<param> ::= <number>|<bound>
<number> ::= an integer (NRf data)
             limits:  0:32767
<bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"status:questionable:frequency:ntransition 0"
                    Output 719;"Stat:Ques:Freq:Ntr 7"
```

Query Syntax: `STATus:QUEStionable:FREQuency:NTRansition?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Questionable Frequency negative transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Questionable Frequency Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of negative transition registers in register sets.

STATus:QUEStionable:FREQuency:PTRansition — command/query

Sets bits in the Questionable Frequency positive transition register.

Command Syntax: STATus:QUEStionable:FREQuency:PTRansition <param>

```
      <param> ::= <number>|<bound>
     <number> ::= an integer (NRf data)
                  limits:  0:32767
      <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"STAT:QUES:FREQ:PTR 2"
                    Output 719;"status:questionable:frequency:ptransition 3"
```

Query Syntax: STATus:QUEStionable:FREQuency:PTRansition?

Return Format: <number>

```
     <number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Questionable Frequency positive transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Questionable Frequency Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of positive transition registers in register sets.

STATus:QUEStionable:LIMit:CONDition? query

Reads the Limit Fail condition register.

Query Syntax: `STATus:QUEStionable:LIMit:CONDition?`

```
Example Statements: Output 719;"stat:ques:lim:cond?"
                    Output 719;"Status:Questionable:Limit:Condition?"
```

Return Format: `<number>`

`<number> ::= an integer (NR1 data)`

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Limit Fail condition register. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Limit Fail Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of condition registers in register sets.

STATus:QUEStionable:LIMit:ENABle — command/query

Sets bits in the Limit Fail enable register.

Command Syntax: `STATus:QUEStionable:LIMit:ENABle {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"STATUS:QUESTIONABLE:LIMIT:ENABLE 4"
                    Output 719;"stat:ques:lim:enab 12"
```

Query Syntax: `STATus:QUEStionable:LIMit:ENABle?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Limit Fail enable register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Limit Fail Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of enable registers in register sets.

STATus:QUEStionable:LIMit[:EVENt]?

query

Reads and clears the Limit Fail event register.

Query Syntax: `STATus:QUEStionable:LIMit[:EVENt]?`

Example Statements:

```
Output 719;"Stat:Ques:Lim:Even?"
Output 719;"STATUS:QUESTIONABLE:LIMIT?"
```

Return Format: <number>

<number> ::= an integer (NR1 data)

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Limit Fail event register. (The decimal weight of a bit is 2^n, where *n* is the bit number.)

Note The Limit Fail event register is automatically cleared after it is read by this query.

See "Limit Fail Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of event registers in register sets.

STATus:QUEStionable:LIMit:NTRansition command/query

Sets bits in the Limit Fail negative transition register.

Command Syntax: `STATus:QUEStionable:LIMit:NTRansition <param>`

```
<param> ::= <number>|<bound>
<number> ::= an integer (NRf data)
             limits: 0:32767
<bound> ::= MAX|MIN
```

Example Statements:
```
Output 719;"status:questionable:limit:ntransition 0"
Output 719;"Stat:Ques:Lim:Ntr 15"
```

Query Syntax: `STATus:QUEStionable:LIMit:NTRansition?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:
```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Limit Fail negative transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Limit Fail Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of negative transition registers in register sets.

STATus:QUEStionable:LIMit:PTRansition **command/query**

Sets bits in the Limit Fail positive transition register.

Command Syntax: `STATus:QUEStionable:LIMit:PTRansition <param>`

```
      <param> ::= <number>|<bound>
     <number> ::= an integer (NRf data)
                  limits:  0:32767
      <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"STAT:QUES:LIM:PTR 2"
                    Output 719;"status:questionable:limit:ptransition 3"
```

Query Syntax: `STATus:QUEStionable:LIMit:PTRansition?`

Return Format: `<number>`

```
     <number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Limit Fail positive transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Limit Fail Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of positive transition registers in register sets.

STATus:QUEStionable:NTRansition — command/query

Sets bits in the Questionable Data negative transition register.

Command Syntax: `STATus:QUEStionable:NTRansition {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767
 <bound> ::= MAX|MIN
```

Example Statements:

```
Output 719;"Status:Questionable:Ntransition 8"
Output 719;"STAT:QUES:NTR 65535"
```

Query Syntax: `STATus:QUEStionable:NTRansition?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Questionable Data negative transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Questionable Data Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of negative transition registers in register sets.

STATus:QUEStionable:POWer:CONDition? — query

Reads the Questionable Power condition register.

Query Syntax: `STATus:QUEStionable:POWer:CONDition?`

Example Statements:

```
Output 719;"stat:ques:pow:cond?"
Output 719;"Status:Questionable:Power:Condition?"
```

Return Format: <number>

<number> ::= an integer (NR1 data)

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Questionable Power condition register. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Questionable Power Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of condition registers in register sets.

STATus:QUEStionable:POWer:ENABle — command/query

Sets bits in the Questionable Power enable register.

Command Syntax: `STATus:QUEStionable:POWer:ENABle {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"STATUS:QUESTIONABLE:POWER:ENABLE 17"
                    Output 719;"stat:ques:pow:enab 31"
```

Query Syntax: `STATus:QUEStionable:POWer:ENABle?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Questionable Power enable register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Questionable Power Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of enable registers in register sets.

STATus:QUEStionable:POWer[:EVENt]? query

Reads and clears the Questionable Power event register.

Query Syntax: `STATus:QUEStionable:POWer[:EVENt]?`

Example Statements:
```
Output 719;"Stat:Ques:Pow?"
Output 719;"STATUS:QUESTIONABLE:POWER:EVENT?"
```

Return Format: <number>

<number> ::= an integer (NR1 data)

Attribute Summary:
Preset state: not applicable
Overlapped: no
Pass control required: no

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the Questionable Power event register. (The decimal weight of a bit is 2^n, where n is the bit number.)

Note

The Questionable Power event register is automatically cleared after it is read by this query.

See "Questionable Power Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of event registers in register sets.

STATus:QUEStionable:POWer:NTRansition command/query

Sets bits in the Questionable Power negative transition register.

Command Syntax: `STATus:QUEStionable:POWer:NTRansition {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits: 0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"status:questionable:power:ntransition 0"
                    Output 719;"Stat:Ques:Pow:Ntr 12"
```

Query Syntax: `STATus:QUEStionable:POWer:NTRansition?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Questionable Power negative transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where *n* is the bit number.)

See "Questionable Power Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of negative transition registers in register sets.

STATus:QUEStionable:POWer:PTRansition

command/query

Sets bits in the Questionable Power positive transition register.

Command Syntax: `STATus:QUEStionable:POWer:PTRansition {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767

 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"STAT:QUES:POW:PTR 65535"
                    Output 719;"status:questionable:power:ptransition 2"
```

Query Syntax: `STATus:QUEStionable:POWer:PTRansition?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Questionable Power positive transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Questionable Power Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of positive transition registers in register sets.

STATus:QUEStionable:PTRansition **command/query**

Sets bits in the Questionable Data positive transition register.

Command Syntax: `STATus:QUEStionable:PTRansition {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767

 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Status:Questionable:Ptransition 21"
                    Output 719;"STAT:QUES:PTR 31"
```

Query Syntax: `STATus:QUEStionable:PTRansition?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the Questionable Data positive transition register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "Questionable Data Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of positive transition registers in register sets.

STATus:USER:ENABle **command/query**

Sets bits in the User Defined enable register.

Command Syntax: `STATus:USER:ENABle {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"stat:user:enab 65535"
                    Output 719;"Status:User:Enable 1023"
```

Query Syntax: `STATus:USER:ENABle?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

To set a single bit in the User Defined enable register to 1, send the bit's decimal weight with this command. To set more than one bit to 1, send the *sum* of the decimal weights of all the bits. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "User Defined Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of enable registers in register sets.

STATus:USER[:EVENt]? query

Reads and clears the User Defined event register.

Query Syntax: `STATus:USER[:EVENt]?`

`Example Statements:` `Output 719;"STATUS:USER:EVENT?"`
`Output 719;"STATUS:USER?"`

Return Format: `<number>`

`<number> ::= an integer (NR1 data)`

Attribute Summary:
`Preset state: not applicable`
`Overlapped: no`
`Pass control required: no`

Description:

This query returns the sum of the decimal weights of all bits currently set to 1 in the User Defined event register. (The decimal weight of a bit is 2^n, where n is the bit number.)

Note The User Defined event register is automatically cleared after it is read by this query.

See "User Defined Register Set" in chapter 5 for a definition of bits in the register set. See "General Status Register Model" in chapter 5 for information about the role of event registers in register sets.

STATus:USER:PULSe

command

Pulse bits in the User Defined condition register.

Command Syntax: `STATus:USER:PULSe {<number>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:32767

 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"Stat:User:Puls 0"
                    Output 719;"STATUS:USER:PULSE 512"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

Each bit in the User Defined condition register is normally set to 0, but is set to 1 (briefly) when you send the bit's decimal weight with the STAT:USER:PULS command. (The decimal weight of a bit is 2^n, where n is the bit number.)

See "User Defined Register Set" in chapter 5 for more information.

28

SYSTem Subsystem

The SYSTem subsystem collects commands that are not related to analyzer performance. Instead, these commands control global functions, such as instrument preset, time, and date.

SYSTem:BEEPer:STATe — command/query

Toggles the analyzer's beeper on and off.

Command Syntax: `SYSTem:BEEPer:STATe {OFF|0|ON|1}`

Example Statements:
```
Output 719;"system:beeper:state on"
Output 719;"Syst:Beep:Stat 1"
```

Query Syntax: `SYSTem:BEEPer:STATe?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: no
Pass control required: no
```

Description:

When the beeper is enabled, it emits an audible tone when some messages are either displayed or placed in the error queue. It also emits an audible tone when a trace falls outside its specified limits if limit testing *and* the limit-fail beeper are enabled (DISP:LIM:STAT ON and DISP:LIM:BEEP ON).

SYSTem:COMMunicate:GPIB:ADDRess — command/query

Sets the analyzer's HP-IB address.

Command Syntax: `SYSTem:COMMunicate:GPIB:ADDRess {<number>|<step>|<bound>}`

```
<number> ::= an integer (NRf data)
             limits:  0:30
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

Example Statements:

```
Output 719;"SYST:COMM:GPIB:ADDR 19"
Output 719;"system:communicate:gpib:address 11"
```

Query Syntax: `SYSTem:COMMunicate:GPIB:ADDRess?`

Return Format: `<number>`

```
<number> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

The analyzer's address is saved in non-volatile memory, so it is retained when you turn the analyzer off and on.

Note

When you use this command, wait at least 5 seconds before sending another command to the new address.

SYSTem:COMMunicate:GPIB:ECHO command/query

Enables and disables the echoing of TMSL command mnemonics to the analyzer's screen.

Command Syntax: `SYSTem:COMMunicate:GPIB:ECHO {OFF|0|ON|1}`

Example Statements:
```
Output 719;"System:Communicate:Gpib:Echo Off"
Output 719;"SYST:COMM:GPIB:ECHO 1"
```

Query Syntax: `SYSTem:COMMunicate:GPIB:ECHO?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +0
Overlapped: no
Pass control required: no
```

Description:

When echoing is enabled, you can operate the analyzer from the front panel and it will display the TMSL commands you must send over the HP-IB to achieve the same results. Mnemonics are displayed in the lower-left corner of the screen.

SYSTem:DATE

command/query

Sets the date in the analyzer's battery-backed clock.

Command Syntax: `SYSTem:DATE <year>,<month>,<day>`

```
<year> ::= MAX|MIN|an integer (NRf data)
           limits:  0:9999
<month> ::= MAX|MIN|an integer (NRf data)
           limits:  1:12
<day> ::= MAX|MIN|an integer (NRf data)
           limits:  1:31
```

```
Example Statements: Output 719;"syst:date 1990,2,20"
                    Output 719;"System:Date 1991,11,4"
```

Query Syntax: `SYSTem:DATE?`

Return Format: `<year>,<month>,<day>`

```
<year> ::= an integer (NR1 data)
<month> ::= an integer (NR1 data)
<day> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

You must enter the year as a four-digit number, including century and millennium information (1990, not 90).

SYSTem:ERRor? query

Returns one error message from the analyzer's error queue.

Query Syntax: `SYSTem:ERRor?`

```
Example Statements: Output 719;"SYSTEM:ERROR?"
                    Output 719;"syst:err?"
```

Return Format: `<err_num>,"<gen_info>;<details>"`

```
 <err_num> ::= an integer (NR1 data)
<gen_info> ::= general description of error
 <details> ::= additional details (if any)
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The error queue temporarily stores up to 20 error messages. When you send the SYST:ERR query, one message is moved from the error queue to the output queue so your controller can read the message. The error queue delivers messages to the output queue in the order received.

Note

The error queue is cleared when you turn on the analyzer and when you send the *CLS command.

Appendix B lists the HP 3588A's error messages.

SYSTem:PRESet — command

Returns most of the analyzer's parameters to their preset states.

Command Syntax: `SYSTem:PRESet`

Example Statements:
```
Output 719;"Syst:Pres"
Output 719;"SYSTEM:PRESET"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

In addition to returning parameters to their preset states, this command does all of the following things:

- Cancels any pending *OPC command or query.
- Clears the error queue.
- Clears all event registers (sets all bits to 0).

Note

The preset state of each parameter is listed under the Attribute Summary of the associated command.

SYST:PRES does not affect the following parameters:

- The state of the Power-on Status Clear flag.
- The state of all enable and transition registers.
- The HP-IB input and output queues.
- The time and date. (SYST:TIME and SYST:DATE).
- The HP-IB address settings. (SYST:COMM:GPIB:ADDR, PLOT:ADDR, and PRIN:ADDR)
- The HP-IB controller capability setting.
- The default disk selection. (MMEM:MSI)
- Contents of limit, data, function, and constant registers.
- Contents of the RAM disks.
- Calibration constants.

SYSTem:RPGLock — command/query

Enables and disables the knob on the analyzer's front panel.

Command Syntax: `SYSTem:RPGLock {OFF|0|ON|1}`

Example Statements:
```
Output 719;"System:Rpglock Off"
Output 719;"SYST:RPGL 1"
```

Query Syntax: `SYSTem:RPGLock?`

Return Format: `+{0|1}`

Attribute Summary:
```
Preset state: +1
Overlapped: no
Pass control required: no
```

Description:

When the knob is enabled, front-panel operators can position the marker or scroll through the disk catalog while the analyzer is being controlled via HP-IB.

SYSTem:SET **command/query**

Transfers an instrument state between the analyzer and an external controller.

Command Syntax: `SYSTem:SET <block>`

```
          <block> ::= #<byte>[<length_bytes>]<data_bytes>
           <byte> ::= one ASCII-encoded byte specifying the number of length
                      bytes to follow
   <length_bytes> ::= ASCII-encoded bytes specifying the number of data
                      bytes to follow
     <data_bytes> ::= the bytes that define an instrument state

Example Statements: Output 719;"system:set?"
                    Output 719;"Syst:Set?"
```

Query Syntax: `SYSTem:SET?`

Return Format: `<block>`

Attribute Summary:

```
Preset state: not defined
Overlapped: yes
Pass control required: no
```

Description:

This command transfers a complete instrument state—the same information contained in a state file—between the analyzer and your controller. This allows you to store an instrument state on your controller's file system. The state cannot be altered.

When you transfer an instrument state to the analyzer, you can use either the definite or the indefinite length block syntax. When the analyzer returns the state to a controller, it always uses the definite length block syntax. See "Block Data" in chapter 4 for more information.

SYSTem:SNUMber? query

Returns the analyzer's serial number.

Query Syntax: `SYSTem:SNUMber?`

```
Example Statements: Output 719;"SYST:SNUM?"
                    Output 719;"system:snumber?"
```

Return Format: `"<ser_num>"`

```
          <ser_num> ::= 10 ASCII characters
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This query returns the analyzer's serial number.

SYSTem:TIME

command/query

Sets the time in the analyzer's battery-backed clock.

Command Syntax: `SYSTem:TIME <hour>,<minute>,<second>`

```
      <hour> ::= MAX|MIN|an integer (NRf data)
                 limits:  0:23
    <minute> ::= MAX|MIN|an integer (NRf data)
                 limits:  0:59
    <second> ::= MAX|MIN|an integer (NRf data)
                 limits:  0:59
```

```
Example Statements: Output 719;"System:Time 14,5,00"
                    Output 719;"SYSTem:TIME 8,58,00"
```

Query Syntax: `SYSTem:TIME?`

Return Format: `<hour>,<minute>,<second>`

```
      <hour> ::= an integer (NR1 data)
    <minute> ::= an integer (NR1 data)
    <second> ::= an integer (NR1 data)
```

Attribute Summary:

```
Preset state: not affected by Preset
Overlapped: no
Pass control required: no
```

Description:

Set the time using a 24-hour format. For example, 3:05 pm becomes 15:05 and is sent as SYST:TIME 15,5,0.

29

TEST Subsystem

Commands in the TEST subsystem access functions that should only be used as described in the *HP 3588A Performance Test Guide*. Refer to that manual for more information on these functions.

TEST:INPut:CONFig command/query

Connects the input circuitry either to the input BNC or the calibrator signal.

Command Syntax: `TEST:INPut:CONFig {FPANel|CALibrator}`

Example Statements:
```
Output 719;"test:inp:conf cal"
Output 719;"Test:Input:Config Fpanel"
```

Query Syntax: `TEST:INPut:CONFigure?`

Return Format: `FPAN|CAL`

Attribute Summary:
```
Preset state: FPAN
Overlapped: no
Pass control required: no
```

Description:

Use this command only as described in the *HP 3588A Performance Test Guide.*

TEST:SOURce:DAC:ATTenuation — command/query

Sets the source's attenuation DAC.

Command Syntax: `TEST:SOURce:DAC:ATTenuation (<value>|<step>|<bound>)`

```
 <value> ::= <number>[DB]
<number> ::= a real number (NRf data)
             limits:  0.0:39.9
  <step> ::= UP|DOWN
 <bound> ::= MAX|MIN
```

```
Example Statements: Output 719;"TEST:SOURCE:DAC:ATTENUATION 0"
                    Output 719;"test:sour:dac:att 20"
```

Query Syntax: `TEST:SOURce:DAC:ATTenuation? [UNIT]`

Return Format: `<number>|("dB")`

```
<number> ::= a real number (NR2 or NR3 data)
```

Attribute Summary:

```
Preset state: +0.00E+0 dB
Overlapped: no
Pass control required: no
```

Description:

Use this command only as described in the *HP 3588A Performance Test Guide*.

30

TRACe Subsystem

The TRACe subsystem contains commands that are used to define and manipulate trace data. One of these commands—TRAC:DATA—allows you to transfer measurement data between the analyzer and an external controller. The following block diagram shows you the position of TRAC:DATA in the data flow:

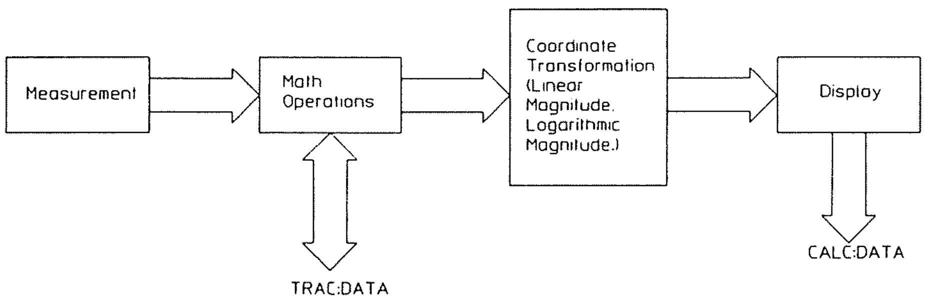


Figure 30-1. Flow of Measurement Data

After measurement data is collected, any specified math operations are performed. Data is then transformed into the specified coordinate system and sent to the display. TRAC:DATA gives you access to the raw measurement data after math operations have been performed. CALC:DATA gives you access to the display data—after the coordinate transformation.

Note You can take measurement data out of the analyzer with either TRACE:DATA or CALC:DATA, but you can only put it back into the analyzer with TRAC:DATA.

The TRACe mnemonic contains an optional trace specifier: [1|2]. To direct a command to trace A, omit the specifier or use 1. To direct a command to trace B, use 2.

TRACe[1|2]:COPY

command

Copies the specified trace into a data register.

Command Syntax: `TRACe[1|2]:COPY {D1|D2|D3|D4|D5|D6|D7|D8}`

Example Statements:

```
Output 719;"Trac:Copy D2"
Output 719;"TRACE1:COPY D5"
```

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

The trace in a data register can either be displayed directly (using DISP:RES) or included in a math function (using CALC:MATH:EXPR).

Note

To save the contents of a data register to disk, you must display the contents in trace A or trace B (with DISP:RES) and then save trace A or trace B (with MMEM:STOR:TRAC).

TRACe[1|2]:DATA **command/query**

Transfers a trace between the analyzer and an external controller.

Command Syntax: `TRACe[1|2]:DATA <block>`

When data is ASCII-encoded, (FORM ASC) <block> takes the following form:

```
<block> ::= [<point>[,<point>]...]
<point> ::= y-axis value for 1 of the 401 points that make up
            a trace
            limits:  -9.9E37:9.9E37
```

When data is binary-encoded, (FORM REAL) <block> takes the following form:

```
        <block> ::= #<byte>[<length_bytes>][<point>]...
         <byte> ::= one ASCII-encoded byte specifying the number of length
                    bytes to follow
<length_bytes> ::= ASCII-encoded bytes specifying the number of data
                    bytes to follow
        <point> ::= y-axis value for 1 of the 401 points that make up
                    a trace
                    limits:  -9.9E37:9.9E37
```

```
Example Statements: Output 719;"trace:data 1,1,1,1,1"
                    Output 719;"Trac2:Data?"
```

Query Syntax: `TRACe[1|2]:DATA?`

Return Format: `<block>`

Attribute Summary:

```
Preset state: not applicable
Overlapped: no
Pass control required: no
```

Description:

This command transfers a block of corrected measurement data (after math operations have been performed) between the analyzer and your controller. The block is transferred as 401 amplitude values. The unit for these values is Vrms.

Note

If you send fewer than 401 values, the analyzer replaces points 0 through *(n - 1)* of the specified trace with the values you send (where *n* is the number of values you send). The remaining points are unchanged.

TRACe[1|2]:TITLe

command/query

Loads a title for the specified trace.

Command Syntax: TRACe[1|2]:TITLe '<title>'

<title> ::= 0 through 15 ASCII characters

Example Statements:
```
Output 719;"TRAC2:TITL 'Filter Shape'"
Output 719;"trace1:title ''"
```

Query Syntax: TRACe[1|2]:TITLe?

Return Format: "<title>"

Attribute Summary:
Preset state: "" (both traces)
Overlapped: no
Pass control required: no

Description:

Trace titles appear above the upper-left corner of traces and can be up to 15 characters long. They replace the default trace titles supplied by the analyzer. You can delete a user-defined trace title by sending the null string ('') with this command; this causes the analyzer to display the default trace title.

Note

If you send *RST or SYST:PRES, user-defined titles are automatically deleted, restoring the default titles for both traces.

31

TRIGger Subsystem

The TRIGger subsystem contains commands that control two of the analyzer's trigger functions. One command selects the source of the trigger signal. The other command triggers the analyzer if the trigger source is BUS. See the ARM subsystem for commands that control the trigger arming functions.

TRIGger[:IMMediate] command

Triggers the analyzer if TRIG:SOUR is BUS.

Command Syntax: `TRIGger[:IMMediate]`

Example Statements:
```
Output 719;"Trigger:Immediate"
Output 719;"TRIG"
```

Attribute Summary:
```
Preset state: not applicable
Overlapped: yes
Pass control required: no
```

Description:

This command triggers the analyzer if the following two conditions are met:

- The HP-IB is designated as the trigger source. (Send the TRIG:SOUR BUS command.)
- The analyzer is waiting to trigger. (Bit 5 of the Standard Operation condition register is set.)

It is ignored at all other times.

TRIG:IMM has the same effect as *TRG. It also has the same effect as the HP-IB bus management command Group Execute Trigger (GET) with the following exception: TRIG:IMM is sent to the input queue and processed in the order received, but GET is processed immediately, even if the input queue contains other commands.

TRIGger:SOURce

command/query

Selects the source of a trigger event.

Command Syntax: `TRIGger:SOURce {IMMediate|EXTernal|BUS}`

Example Statements:
```
Output 719;"trig:sour ext"
Output 719;"Trigger:Source Immediate"
```

Query Syntax: `TRIGger:SOURce?`

Return Format: `IMM|EXT|BUS`

Attribute Summary:
```
Preset state: IMM
Overlapped: yes
Pass control required: no
```

Description:

Send IMM to select free run triggering, which automatically triggers the analyzer as soon as it is armed.

Send EXT to select the analyzer's EXT TRIG connector (on the rear panel) as the trigger source. If EXT is selected, the analyzer is triggered by a high-to-low transition of the TTL signal applied to this connector.

Send BUS to select the analyzer's HP-IB connector (also on the rear panel) as the trigger source. If BUS is selected, the analyzer is triggered when you send any of the following HP-IB commands:

- *TRG
- TRIG:IMM
- Group Execute Trigger (GET)

GET is a *bus management* command. See "Response to Bus Management Commands" in chapter 2 for more information.

Note

The analyzer must be waiting to trigger when it receives an external trigger signal or bus trigger command, otherwise the signal or command is ignored. Bit 5 of the Standard Operation condition register is set to 1 when the analyzer is waiting to trigger.

HP 3588A Command Summary

Introduction

This appendix contains a listing of all the TMSL commands recognized by the HP 3588A. It begins with the *common commands* and then lists the *subsystem commands* in alphabetical order. The appendix uses the following conventions:

- Syntax is taken from the command descriptions in the Command Reference chapters. Refer to those descriptions for more information on a particular command.
- Indentation is used to show a subsystem command's relative position in the TMSL command tree. See chapter 3 for more information on the command tree.
- An italic font is used for commands that fill a position in the command tree but do not affect the analyzer state. For example, DISPlay[1|2]:LIMit:UPPer gives you access to the command DISPlay[1|2]:LIMit:UPPer:DELete, but it does not affect the analyzer state. (In fact, it is not even recognized by the command parser.)

Command List

Common Commands

*CAL?
*CLS
*ESE {<number> | <bound>}
*ESR?
*IDN?
*OPC
*PCB <number_1>[,<number_2>]
*PSC {<number> | <bound>}
*RST
*SRE {<number> | <bound>}
*STB?
*TRG
*TST?
*WAI

ABORt

ARM

ARM[:IMMediate]
ARM:SOURce {IMMediate|MANual}

AVERage

AVERage:COUNt {<number> | <step> | <bound>}
AVERage[:STATe] {OFF|0|ON|1}
AVERage:TCONtrol {EXPonential}
AVERage:TYPE {RMS|MAX|VIDeo|PEAK}

CALCulate[1|2]

CALCulate[1|2]:DATA?
CALCulate[1|2]:FORMat {MLINear|MLOGarithmic|NONE}
CALCulate[1|2]:MATH
CALCulate[1|2]:MATH:CONStant <const>,{<number> | <bound>}
CALCulate[1|2]:MATH:DATA <block>
CALCulate[1|2]:MATH[:EXPRession] <func>,<expr>

CALibration

CALibration[:ALL]?
CALibration:AUTO {OFF|0|ON|1|ONCE}
CALibration:CORRection
CALibration:CORRection:SRATe {OFF|0|ON|1}

DIAGnostics

DIAGnostics:SOURce
DIAGnostics:SOURce:PAD
DIAGnostics:SOURce:PAD:TEN {IN|OUT}
DIAGnostics:SOURce:PADA
DIAGnostics:SOURce:PADA:TWENty {IN|OUT}
DIAGnostics:SOURce:PADB
DIAGnostics:SOURce:PADB:TWENty {IN|OUT}

DISPlay[1|2]

DISPlay[1|2]:LIMit
DISPlay[1|2]:LIMit:BEEP {OFF|0|ON|1}
DISPlay[1|2]:LIMit:LINE {OFF|0|ON|1}
DISPlay[1|2]:LIMit:LOWer
DISPlay[1|2]:LIMit:LOWer:DATA <block>
DISPlay[1|2]:LIMit:LOWer:DELete
DISPlay[1|2]:LIMit:LOWer:MOVE {<number>|<step>|<bound>}
DISPlay[1|2]:LIMit:LOWer:REPort?
DISPlay[1|2]:LIMit:LOWer:SEGMent <block>
DISPlay[1|2]:LIMit:MODE {ABSolute|RELative}
DISPlay[1|2]:LIMit:RESult?
DISPlay[1|2]:LIMit:STATe {OFF|0|ON|1}
DISPlay[1|2]:LIMit:UPPer
DISPlay[1|2]:LIMit:UPPer:DATA <block>
DISPlay[1|2]:LIMit:UPPer:DELete
DISPlay[1|2]:LIMit:UPPer:MOVE {<number>|<step>|<bound>}
DISPlay[1|2]:LIMit:UPPer:REPort?
DISPlay[1|2]:LIMit:UPPer:SEGMent <block>
DISPlay[1|2]:PARTition {OFF|0|FULL|UPPer|LOWer}
DISPlay[1|2]:PARTition:CLEar
DISPlay[1|2]:RESults <param>
DISPlay[1|2]:Y
DISPlay[1|2]:Y:SCALe
DISPlay[1|2]:Y:SCALe:AUTO {ONCE|OFF|0}
DISPlay[1|2]:Y:SCALe:MAXimum {<value>|<step>|<bound>}
DISPlay[1|2]:Y:SCALe:MAXimum:AUTO {OFF|0|ON|1}
DISPlay[1|2]:Y:SCALe:PDIVision {<value>|<step>|<bound>}

FORMat

FORMat[:DATA] {ASCii|REAL}[,{<number>|<bound>}]

INITiate

INITiate:CONTinuous {ON|1}
INITiate[:IMMediate]

INPut

INPut:IMPedance {<value>|<bound>}
INPut:IMPedance:REFerence {<value>|<step>|<bound>}
INPut:TRIP
INPut:TRIP:CLEar

MARKer[1|2]

MARKer[1|2]:FUNCtion
MARKer[1|2]:FUNCtion:FCOunt {OFF|0|ON|1}
MARKer[1|2]:FUNCtion:NOISe {OFF|0|ON|1}
MARKer[1|2]:MAXimum
MARKer[1|2]:MAXimum:GLOBal
MARKer[1|2]:MAXimum:LEFT
MARKer[1|2]:MAXimum:RIGHt
MARKer[1|2]:MAXimum:TRACk {OFF|0|ON|1}
MARKer[1|2]:MINimum
MARKer[1|2]:MINimum:GLOBal
MARKer[1|2]:OFFSet {OFF|0|ON|1}
MARKer[1|2]:OFFSet:DELTa
MARKer[1|2]:OFFSet:DELTa:X {<value>|<bound>}
MARKer[1|2]:OFFSet:DELTa:Y {<value>|<bound>}
MARKer[1|2]:OFFSet:X {<value>|<step>|<bound>}
MARKer[1|2]:OFFSet:Y {<value>|<bound>}
MARKer[1|2]:POINt {<number>|<step>|<bound>}
MARKer[1|2][:STATe] {OFF|0|ON|1}
MARKer[1|2]:TO
MARKer[1|2]:TO:SPAN
MARKer[1|2]:X {<value>|<step>|<bound>}
MARKer[1|2]:X:FCOunt? [UNIT]
MARKer[1|2]:Y? [UNIT]
MARKer[1|2]:Y:NOISe? [UNIT]

MMEMory

- MMEMory:COPY '<pathname>','<pathname>'
- MMEMory:DELete '[<disk>][<filename>]'
- *MMEMory:GET*
  - MMEMory:GET:PROGram '[<disk>]<filename>'
- MMEMory:INITialize '<disk>'[,{<number> | <bound>}]
- *MMEMory:LOAD*
  - *MMEMory:LOAD:LIMit*
    - MMEMory:LOAD:LIMit:LOWer {A|B},'[<disk>]<filename>'
    - MMEMory:LOAD:LIMit:UPPer {A|B},'[<disk>]<filename>'
  - MMEMory:LOAD:MATH '[<disk>]<filename>'
  - MMEMory:LOAD:PROGram '[<disk>]<filename>'
  - MMEMory:LOAD:STATe {1|MAX|MIN},'[<disk>]<filename>'
  - MMEMory:LOAD:TRACe <data_reg>,'[<disk>]<filename>'
- MMEMory:MSI '<disk>'
- MMEMory:PACK ['<disk>']
- MMEMory:REName '<pathname>','<filename>'
- *MMEMory:RESave*
  - MMEMory:RESave:PROGram '[<disk>]<filename>'
- *MMEMory:SAVE*
  - MMEMory:SAVE:PROGram '[<disk>]<filename>'
- *MMEMory:STORe*
  - *MMEMory:STORe:LIMit*
    - MMEMory:STORe:LIMit:LOWer {A|B},'[<disk>]<filename>'
    - MMEMory:STORe:LIMit:UPPer {A|B},'[<disk>]<filename>'
  - MMEMory:STORe:MATH '[<disk>]<filename>'
  - MMEMory:STORe:PROGram '[<disk>]<filename>'
  - MMEMory:STORe:STATe {1|MAX|MIN},'[<disk>]<filename>'
  - MMEMory:STORe:TRACe {A|B},'[<disk>]<filename>'

PLOT

PLOT:ADDRess {<number> | <step> | <bound>}
PLOT:DUMP
PLOT:DUMP:ALL
PLOT:DUMP:GRATicule
PLOT:DUMP:MARKer
PLOT:DUMP:OFFSet
PLOT:DUMP:OFFSet:MARKer
PLOT:DUMP:TRACe
PLOT:EJECt {OFF|0|ON|1}
PLOT:LTYPe
PLOT:LTYPe:TRACe[1|2] {<number> | <bound>}
PLOT:PEN
PLOT:PEN:ALPHa {<number> | <step> | <bound>}
PLOT:PEN:GRATicule {<number> | <step> | <bound>}
PLOT:PEN:INITialize
PLOT:PEN:MARKer[1|2] {<number> | <step> | <bound>}
PLOT:PEN:TRACe[1|2] {<number> | <step> | <bound>}
PLOT:SPEed {<number> | <step> | <bound>}

PRINt

PRINt:ADDRess {<number> | <step> | <bound>}
PRINt:DUMP
PRINt:DUMP:ALL

PROGram

PROGram[:SELected]
PROGram[:SELected]:DEFine <block>
PROGram[:SELected]:DELete
PROGram[:SELected]:DELete[:SELected]
PROGram[:SELected]:MALLocate {<number> | <bound> | DEFault}
PROGram[:SELected]:NUMBer '<variable>',<block>
PROGram[:SELected]:STATe <param>
PROGram[:SELected]:STRing '<variable>','<string>'

SCReen

SCReen:ACTive {A|B}
SCReen:ANNotation {OFF|0|ON|1}
SCReen:CONTents {TRACe|STATe|MMEMory}
SCReen:FORMat {SINGle|ULOWer|FBACk}
SCReen:GRATicule {OFF|0|ON|1}
SCReen[:STATe] {OFF|0|ON|1}

[SENSe:]

- *[SENSe:]BANDwidth*
  - [SENSe:]BANDwidth:NOISe?
    - [SENSe:]BANDwidth:NOISe:CORRection?
  - [SENSe:]BANDwidth[:RESolution] {<value>|<step>|<bound>}
    - [SENSe:]BANDwidth[:RESolution]:AUTO {OFF|0|ON|1|ONCE}
    - [SENSe:]BANDwidth[:RESolution]:FFT?
  - [SENSe:]BANDwidth:VIDeo {<value>|<step>|<bound>}
    - [SENSe:]BANDwidth:VIDeo:STATe {OFF|0|ON|1}
- *[SENSe:]DETector*
  - [SENSe:]DETector[:FUNCtion] {POSitive|SAMPle}
  - [SENSe:]DETector:STIMe {<value>|<step>|<bound>}
- *[SENSe:]FREQuency*
  - [SENSe:]FREQuency:CENTer {<value>|<step>|<bound>}
    - [SENSe:]FREQuency:CENTer:TRACk {OFF|0|ON|1}
  - [SENSe:]FREQuency:MANual {<value>|<step>|<bound>}
  - [SENSe:]FREQuency:SPAN {<value>|<step>|<bound>}
    - [SENSe:]FREQuency:SPAN:FULL
  - [SENSe:]FREQuency:STARt {<value>|<step>|<bound>}
  - [SENSe:]FREQuency:STEP {<value>|<step>|<bound>}
    - [SENSe:]FREQuency:STEP:STATe {AUTO|MANual}
  - [SENSe:]FREQuency:STOP {<value>|<step>|<bound>}
- [SENSe:]FUNCtion '{POWer:FFT|POWer:SWEPt}'
  - *[SENSe:]FUNCtion:POWer*
    - [SENSe:]FUNCtion:POWer:FFT
    - [SENSe:]FUNCtion:POWer:SWEPt
- *[SENSe:]POWer*
  - [SENSe:]POWer:RANGe {<value>|<step>|<bound>}
    - [SENSe:]POWer:RANGe:AUTO {OFF|0|ON|1|ONCE}
    - [SENSe:]POWer:RANGe:LDIStortion {OFF|0|ON|1}
- [SENSe:]RESTart
- *[SENSe:]SWEep*
  - [SENSe:]SWEep:MODE {AUTO|MANual}
  - [SENSe:]SWEep:TIME {<value>|<step>|<bound>}
- *[SENSe:]WINDow*
  - [SENSe:]WINDow[:TYPE] {HANNing|FLATtop}

SOURce

SOURce:OUTPut
SOURce:OUTPut:IMPedance {<value> | <bound>}
SOURce:OUTPut:IMPedance:MODE {OFF|0|ON|1}
SOURce:OUTPut:PROTection
SOURce:OUTPut:PROTection:CLEar
SOURce:OUTPut[:STATe] {OFF|0|ON|1}
SOURce:POWer
SOURce:POWer[:LEVel]
SOURce:POWer[:LEVel][:IMMediate]
SOURce:POWer[:LEVel][:IMMediate][:AMPLitude] <param>
SOURce:POWer[:LEVel][:IMMediate][:AMPLitude]:STEP <param>

STATus

STATus:DEVice
STATus:DEVice:CONDition?
STATus:DEVice:ENABle {<number> | <bound>}
STATus:DEVice[:EVENt]?
STATus:DEVice:NTRansition {<number> | <bound>}
STATus:DEVice:PTRansition {<number> | <bound>}
STATus:OPERation
STATus:OPERation:CONDition?
STATus:OPERation:ENABle {<number> | <bound>}
STATus:OPERation[:EVENt]?
STATus:OPERation:NTRansition {<number> | <bound>}
STATus:OPERation:PTRansition {<number> | <bound>}
STATus:PRESet
STATus:QUEStionable
STATus:QUEStionable:CONDition?
STATus:QUEStionable:ENABle {<number> | <bound>}
STATus:QUEStionable[:EVENt]?
STATus:QUEStionable:FREQuency
STATus:QUEStionable:FREQuency:CONDition?
STATus:QUEStionable:FREQuency:ENABle {<number> | <bound>}
STATus:QUEStionable:FREQuency[:EVENt]?
STATus:QUEStionable:FREQuency:NTRansition <param>
STATus:QUEStionable:FREQuency:PTRansition <param>
STATus:QUEStionable:LIMit
STATus:QUEStionable:LIMit:CONDition?
STATus:QUEStionable:LIMit:ENABle {<number> | <bound>}
STATus:QUEStionable:LIMit[:EVENt]?
STATus:QUEStionable:LIMit:NTRansition <param>
STATus:QUEStionable:LIMit:PTRansition <param>
STATus:QUEStionable:NTRansition {<number> | <bound>}
STATus:QUEStionable:POWer
STATus:QUEStionable:POWer:CONDition?
STATus:QUEStionable:POWer:ENABle {<number> | <bound>}
STATus:QUEStionable:POWer[:EVENt]?
STATus:QUEStionable:POWer:NTRansition {<number> | <bound>}
STATus:QUEStionable:POWer:PTRansition {<number> | <bound>}
STATus:QUEStionable:PTRansition {<number> | <bound>}
STATus:USER
STATus:USER:ENABle {<number> | <bound>}
STATus:USER[:EVENt]?
STATus:USER:PULSe {<number> | <bound>}

SYSTem

SYSTem:BEEPer
SYSTem:BEEPer:STATe {OFF|0|ON|1}
SYSTem:COMMunicate
SYSTem:COMMunicate:GPIB
SYSTem:COMMunicate:GPIB:ADDRess {<number>|<step>|<bound>}
SYSTem:COMMunicate:GPIB:ECHO {OFF|0|ON|1}
SYSTem:DATE <year>,<month>,<day>
SYSTem:ERRor?
SYSTem:PRESet
SYSTem:RPGLock {OFF|0|ON|1}
SYSTem:SET <block>
SYSTem:SNUMber?
SYSTem:TIME <hour>,<minute>,<second>

TEST

TEST:INPut
TEST:INPut:CONFigure {FPANel|CALibrator}
TEST:SOURce
TEST:SOURce:DAC
TEST:SOURce:DAC:ATTenuation {<value>|<step>|<bound>}

TRACe[1|2]

TRACe[1|2]:COPY {D1|D2|D3|D4|D5|D6|D7|D8}
TRACe[1|2]:DATA <block>
TRACe[1|2]:TITLe '<title>'

TRIGger

TRIGger[:IMMediate]
TRIGger:SOURce {IMMediate|EXTernal|BUS}

B

Error Messages

Introduction

This appendix contains a listing of all the error messages that can be generated by the HP 3588A in response to TMSL commands. Each message consists of an error number (always negative) followed by a string. The string contains a general description of the error followed by additional information about the cause of the error. In many cases, the additional information is just a listing of the command that caused the error.

In this appendix, error numbers and their general descriptions are shown using a bold font. Phrases that complete the descriptions with additional information are grouped under the associated error number.

Up to 20 error messages are temporarily stored in the analyzer's error queue. They are returned to the controller, one message at a time, when you send the SYST:ERR query.

Command Errors

-100,"Command error;

-101,"Invalid character;

-102,"Syntax error;

-103,"Invalid separator;

-104,"Data type error;

-105,"GET not allowed;

-108,"Parameter not allowed;
Too many parameters.

-109,"Missing parameter;
Missing Parameter
Extra Parameter(s)

-110,"Command header error;

-111,"Header separator error;

-112,"Program mnemonic too long;

-113,"Undefined header;

-114,"Header suffix out of range;

-120,"Numeric data error;

-121,"Invalid character in number;

-123,"Exponent too large;

-124,"Too many digits;

-128,"Numeric data not allowed;

-130,"Suffix error;

-131,"Invalid suffix;
Invalid Suffix

-138,"Suffix not allowed;

-140,"Character data error;

-141,"Invalid character data;

-148,"Character data not allowed;

-150,"String data error;

-151,"Invalid string data;

-158,"String data not allowed;

-160,"Block data error;

-161,"Invalid block data;

-168,"Block data not allowed;

-170,"Expression error;

-171,"Invalid expression;

-178,"Expression data not allowed;

Execution Errors

-200,"Execution error;

Recording mode canceled because: %s
Instrument BASIC not installed.
SAVE/RECALL PROGRAM Not Allowed while RECORDING ENABLED.
Marker Value is not valid.
Received HP-IB control without requesting it.
HP-IB control not received.
Instrument State Controller Missing
Invalid Function Code
Invalid Instrument State Request
Invalid Instrument State Value
Invalid Instrument State
Plot/Print Already In Progress
Serial number must be 10 characters.
Not a valid serial number.
F%d definition is not valid for execution.

-201,"Invalid while in local;

-210,"Trigger error;

-211,"Trigger ignored;

-212,"Arm ignored;

-220,"Parameter error;

Invalid Instrument State Parameter

-221,"Settings conflict;

Invalid program state change requested.
Marker is not ON.
Offset Marker is not ON.
Invalid during NARROW BAND ZOOM.

-222,"Data out of range;

Out of range.

-223,"Too much data;

-224,"Illegal parameter value;

-231,"Data questionable;

-240,"Hardware error;

Hardware failure: %s
INPUT PROTECTION TRIPPED Clear trip in Range menu.
SOURCE PROTECTION TRIPPED Clear trip in Source menu.
Local Oscillator Unlocked
Hardware Failure

-241,"Hardware missing;

-250,"Mass storage error;

Improper mass storage unit specifier.
Improper file name
Disk operation aborted.
Mass storage units must be same when renaming.
Improper file type.
File does not contain a STATE.
File does not contain a TRACE.
File does not contain MATH definitions.
File does not contain LIMIT definitions.
Source and destination units are same.
Operation not allowed while file(s) open.
Illegal format parameter(s).
Bad disk.

-251,"Missing mass storage;

Mass storage unit not present.

-252,"Missing media;

Disk not in drive.

-253,"Corrupt media;

Not a valid directory.

-254,"Media full;

Insufficient disk space.

-255,"Directory full;

Full directory.

-256,"File name not found;

File name is undefined.

-257,"File name error;

Duplicate file name.

-258,"Media protected;
Write protected disk.

-260,"Expression error;

-280,"Program error;
%s

-283,"Illegal variable name;
Illegal variable name.

-284,"Program currently running;
Program currently running.

-285,"Program syntax error;
ERROR 949 Syntax error at cursor
Downloaded program line must have a line number.

-286,"Program runtime error;

Device-Specific Errors

-300,"Device-specific error;

-310,"System error;
Calibration Failure
Calibration DMA Timeout
Calibration Overloads
Serial number already set.

-311,"Memory error;

-314,"Save/recall memory lost;

-350,"Too many errors;

Query Errors

-400,"Query error;

-410,"Query INTERRUPTED;

-420,"Query UNTERMINATED;

-430,"Query DEADLOCKED;

-450,"Query not allowed;

Index

Hewlett-Packard Sales and Service Offices

To obtain Servicing information or to order replacement parts, contact the nearest Hewlett-Packard Sales and Service Office listed in HP Catalog, or contact the nearest regional office listed below:

In the United States

California
P.O. Box 4230
1421 South Manhattan Avenue
Fullerton 92631

Georgia
P.O. Box 105005
2000 South Park Place
Atlanta 30339

Illinois
5201 Tollview Drive
Rolling Meadows 60008

New Jersey
W. 120 Century Road
Paramus 07652

In Canada
Hewlett-Packard (Canada) Ltd.
17500 South Service Road
Trans-Canada Highway
Kirkland, Quebec H9J 2M5

In France
Hewlett-Packard France
F-91947 Les Ulis Cedex
Orsay

In German Federal Republic
Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D-6000 Frankfurt 56

In Great Britain
Hewlett-Packard Ltd.
King Street Lane
Winnersh, Wokingham
Berkshire RG11 5AR

In Other European Countries
Switzerland
Hewlett-Packard (Schweiz) AG
7, rue du Bois-du-Lan
Case Postale 365
CH-1217 Meyrin

In All Other Locations
Hewlett-Packard Inter-Americas
3155 Porter Drive
Palo Alto, California 94304

HEWLETT PACKARD

Printed in U.S.A.

HEWLETT
PACKARD

9282-1078
Binder Only Part No.

Made in U.S.A.

HEWLETT
PACKARD

HP 3588A HP-IB Programming Reference

Printed in U.S.A.